HT48R05A-1/HT48R06A-1
Cost-Effective I/O 型单片机
使用手册

目录

前言

自从盛群半导体公司成立以来，既致力于单片机产品的设计与开发。虽然盛群半导体提供给客户各式各样的半导体芯片，但其中单片机仍是盛群的主要关键产品，未来盛群半导体仍将继续扩展单片机产品系列的完整性与功能性。通过长期累积的单片机研发经验与技术，盛群半导体能为各式各样的应用范围开发出高性能且低价位的单片机芯片。盛群的 Cost-Effective I/O 型单片机提供客户绝佳的产品方案，大大地为顾客提升他们产品的功能，当设计者使用盛群所开发出的各式开发工具时，更可减少产品开发周期并大大的增加他们的产品附加价值。

为了使用者阅读方便，本手册分成三部份。关于一般的单片机的规格信息可在第一部份中找到。与单片机程序相关的信息，如指令集、指令定义和汇编语言编译伪指令，可在第二部份找到。第三部份则是关于盛群半导体的开发工具有关如何安装和使用的相关信息。

希望使用 Cost-Effective I/O 型单片机的盛群半导体客户，通过这本手册，能以一种简单、有效、且完整的方法，实现他们在单片机上的各种应用。由于盛群半导体将单片机规格、程序规划和开发工具等信息结合在一本使用手册上，预期客户将可充分利用盛群半导体各种单片机的特色，获取最大的产品优势。盛群半导体也欢迎客户提供宝贵的意见和建议，以作为我们未来的改进参考。

第一部份

单片机概论

第一章

硬件结构

1

本章主要为 Cost-Effective I/O 型单片机的规格信息，并且包含了所有参数和相关的硬件信息，这些信息提供设计者此类单片机的主要硬件特性细节，结合程序部份的信息将能够让使用者快速且成功地实现各种单片机的应用。参考本章中的相关部份，也可保证使用者可以充分利用 Cost-Effective I/O 型单片机。

简介

HT48R05A-1/HT48C05 和 HT48R06A-1/HT48C06 是 8 位高性能、高效益的 RISC 结构单片机，适用于多输入/输出控制产品。其内部特性，如暂停、唤醒功能、振荡器选择和蜂鸣器驱动等，提升了单片机的灵活度，而这些特性也同时保证实际应用时只需要最少的外部器件，进而降低了整个产品成本。有了低功耗、高性能、灵活控制的输入/输出口和低成本等优势，该系列单片机适合被广泛应用在如工业控制、消费性产品和子系统控制器等场合。该系列所有的单片机都拥有相同的功能和特性，主要的不同在于数据存储器和程序存储器的容量的大小。

HT48R05A-1 和 HT48R06A-1 两者都是属于一次可编程(One-Time Programmable, OTP)单片机，当配合使用盛群半导体的程序开发工具时，可简单有效的更新程序。这提供了设计者快速有效的开发途径。而对于那些已经设计成熟的应用，掩膜版的 HT48C05 和 HT48C06 则可满足大量生产和低成本的需求。由于和 OTP 版的功能完全兼容，HT48C05 和 HT48C06 对于已经设计完成而想要降低成本的产品，提供了一个理想的解决方案。

特性

技术特性

- 高性能 RISC 结构
- 低功率完全静态 CMOS 设计
- 工作电压：
  在 4MHz 下，由 2.2V 到 5.5V
  在 8MHz 下，由 3.3V 到 5.5V
- 功率损耗：
  在 5V/4MHz 下，典型值为 2mA
  不使用看门狗定时器时，静态(standby)电流小于 1μA
- 温度范围：
  工作温度–40°C 到 85°C(工业级规格)
  储存温度–50°C 到 125°C

内核特性

- 程序存储器
  0.5K×14 OTP/Mask ROM (HT48R05A-1/HT48C05)
  1K×14 OTP/Mask ROM (HT48R06A-1/HT48C06)
- 数据存储器
  32×8 SRAM (HT48R05A-1/HT48C05)
  64×8 SRAM (HT48R06A-1/HT48C06)
- 14 位数据查表
- 两级硬件堆栈结构
- 直接和间接数据寻址模式
- 位操作指令
- 63 条功能强大的指令
- 大多数指令执行时间只需要一个指令周期

周边特性

- 提供最多 13 个具有上拉功能的双向输入输出口
- PA 口具有唤醒功能
- 外部中断输入
- 事件计数输入
- 具 8 级预分频器(Prescaler)及中断功能的 8 位定时器
- 看门狗定时器(WDT)
- 暂停与唤醒特性可以节省功耗
- PFD/蜂鸣器驱动输出
- 芯片内置晶体及电阻电容振荡电路
- 具有低电压复位特性
- 具有烧录电路接口及程序码保护功能
- HT48C05 和 HT48C06 适用于低价位大量生产
- 16-pin SSOP(237×193mil)，18-pin DIP/SOP 的封装
- 高效完备的软硬件支持工具

选择表

Cost-Effective I/O 型单片机拥有多样的功能特性，其中有些是普通的，有些则是独有的。大部份特性对该系列所有的单片机来说是共有的，主要的区别在于程序存储器和数据存储器的容量大小。为了帮助使用者在应用时能选择适当的单片机，以下的表格提供了各个单片机主要的特性概述。

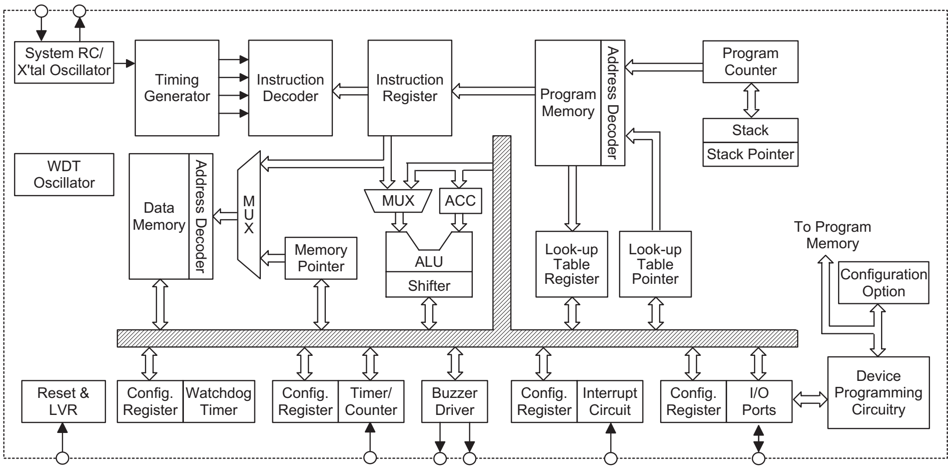
型号	电源	程序存储器	数据存储器	输入/输出口	中断
HT48R05A-1 HT48C05	2.2V~5.5V	0.5K×14	32×8	13	2
HT48R06A-1 HT48C06	2.2V~5.5V	1K×14	64×8	13	2

注意：1. 型号部份包含“C”的为掩膜版本，而“R”则是 OTP 版本。

2. 电压在 2.7V~3.3V 时有低电压复位功能，当不使用此功能时，则工作电压更可降至 2.2V。

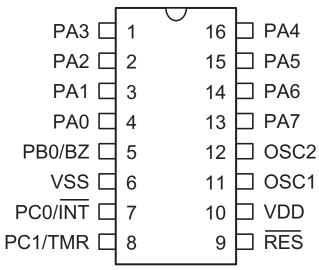
系统框线图

以下系统框线图为 Cost-Effective I/O 型单片机系列的主要功能模块。

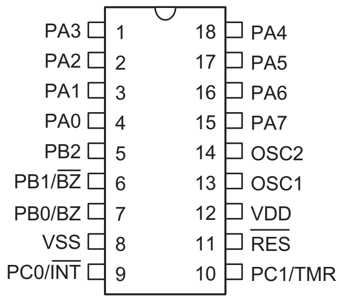


注意：本系统框线图为 OTP 单片机，至于掩膜型单片机则没有 Device Programming Circuitry。

引脚分配



HT48R05A-1/HT48C05
HT48R06A-1/HT48C06
16 SSOP-A



HT48R05A-1/HT48C05
HT48R06A-1/HT48C06
18 DIP-A/18 SOP-A

引脚说明

引脚名称	I/O	掩膜选项	说明
PA0~PA7	I/O	Pull-high Wake-up	8 位双向输入/输出口，每位可由掩膜选项设置成唤醒输入。软件指令决定引脚是 CMOS 输出或 Schmitt Trigger 输入。掩膜选项决定所有引脚是否有上拉电阻。
PB0/BZ PB1/$\overline{\text{BZ}}$ PB2	I/O	Pull-high I/O or BZ/$\overline{\text{BZ}}$	3 位双向输入/输出口。软件指令决定引脚是 CMOS 输出或 Schmitt Trigger 输入。掩膜选项决定所有引脚是否有上拉电阻。PB0 和 PB1 分别与 BZ 和 $\overline{\text{BZ}}$ 引脚共用。
PC0/$\overline{\text{INT}}$ PC1/TMR	I/O	Pull-high	2 位双向输入/输出口。软件指令决定引脚是 CMOS 输出或 Schmitt Trigger 输入。掩膜选项决定所有引脚是否有上拉电阻。外部中断和定时器输入分别与 PC0 和 PC1 引脚共用。外部中断在高电平转低电平时被激发。
OSC1 OSC2	I O	Crystal or RC	OSC1、OSC2 连接 RC 电路或晶体振荡器(由掩膜选项决定)来作为系统时钟。在 RC 模式下，OSC2 的输出端信号是四分之一的系统时钟。
$\overline{\text{RES}}$	I	—	触发复位输入，低电平有效。
VDD	—	—	正电源。
VSS	—	—	负电源，接地。

注意： 1. PA 上的每个引脚可通过掩膜选项被设定成拥有唤醒功能。

2. 单独的引脚或口不可以被选择为拥有上拉电阻，如果选择了上拉配置，则所有输入引脚都将被连接到上拉电阻。

极限参数

供应电压.................................... V_{SS}−0.3V to V_{SS}+6.0V

输入电压.................................... V_{SS}−0.3V to V_{DD}+0.3V

储存温度.................................... −50°C~125°C

工作温度.................................... −40°C~85°C

这里只强调额定功率，超过极限参数功率的范围将对芯片造成损害，芯片在所标示范围外的表现并不能预期，而长期工作在标示范围外条件下也可能影响芯片的可靠性。

直流电气特性

Ta=25°C

符号	参数	测试条件		最小	典型	最大	单位
		V_{DD}	条件				
V_{DD}	Operating Voltage	—	f_{SYS}=4MHz	2.2	—	5.5	V
			f_{SYS}=8MHz	3.3	—	5.5	V
I_{DD1}	Operating Current (Crystal OSC)	3V	No load, f_{SYS}=4MHz	—	0.6	1.5	mA
		5V		—	2	4	mA
I_{DD2}	Operating Current (RC OSC)	3V	No load, f_{SYS}=4MHz	—	0.8	1.5	mA
		5V		—	2.5	4	mA
I_{DD3}	Operating Current (Crystal OSC)	5V	No load, f_{SYS}=8MHz	—	3	5	mA
I_{STB1}	Standby Current (WDT Enabled)	3V	No load, system HALT	—	—	5	μA
		5V		—	—	10	μA
I_{STB2}	Standby Current (WDT Disabled)	3V	No load, system HALT	—	—	1	μA
		5V		—	—	2	μA
V_{IL1}	Input Low Voltage for I/O ports, TMR and $\overline{INT}$	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O ports, TMR and $\overline{INT}$	—	—	$0.7V_{DD}$	—	V_{DD}	V
V_{IL2}	Input Low Voltage ($\overline{RES}$)	—	—	0	—	$0.4V_{DD}$	V
V_{IH2}	Input High Voltage ($\overline{RES}$)	—	—	$0.9V_{DD}$	—	V_{DD}	V
V_{LVR}	Low Voltage Reset	—	LVR enabled	2.7	3.0	3.3	V
I_{OL}	I/O Port Sink Current	3V	$V_{OL}=0.1V_{DD}$	4	8	—	mA
		5V	$V_{OL}=0.1V_{DD}$	10	20	—	mA
I_{OH}	I/O Port Source Current	3V	$V_{OH}=0.9V_{DD}$	–2	–4	—	mA
		5V	$V_{OH}=0.9V_{DD}$	–5	–10	—	mA
R_{PH}	Pull-high Resistance	3V	—	40	60	80	kΩ
		5V	—	10	30	50	kΩ

交流电气特性

Ta=25°C

符号	参数	测试条件		最小	典型	最大	单位
		V_{DD}	条件				
f_{SYS1}	System Clock (Crystal OSC)	—	2.2V ~ 5.5V	400	—	4000	kHz
		—	3.3V ~ 5.5V	400	—	8000	kHz
f_{SYS2}	System Clock (RC OSC)	—	2.2V ~ 5.5V	400	—	4000	kHz
		—	3.3V ~ 5.5V	400	—	8000	kHz
f_{TIMER}	Timer I/P Frequency (TMR)	—	2.2V ~ 5.5V	0	—	4000	kHz
		—	3.3V ~ 5.5V	0	—	8000	kHz
t_{WDTOSC}	Watchdog Oscillator Period	3V	—	45	90	180	μs
		5V	—	32	65	130	μs
t_{WDT1}	Watchdog Time-out Period (RC)	3V	Without WDT Prescaler	11	23	46	ms
		5V		8	17	33	ms
t_{WDT2}	Watchdog Time-out Period (System Clock)	—	Without WDT Prescaler	—	1024	—	t_{SYS}*
t_{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t_{SST}	System Start-up Timer Period	—	Wake-up from HALT	—	1024	—	t_{SYS}*
t_{LVR}	Low Voltage Width to Reset	—	—	1	—	—	ms
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs

* $t_{SYS}=1/f_{SYS1}$ 或 $1/f_{SYS2}$

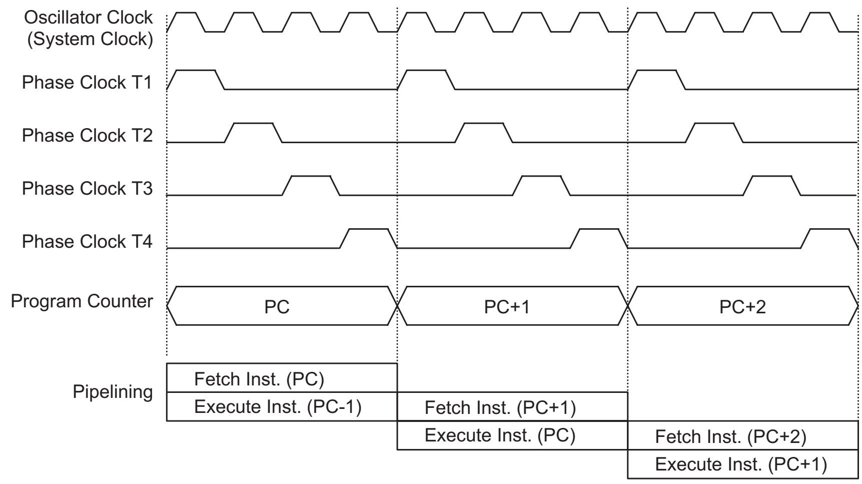
系统结构

内部系统结构是盛群半导体公司 Cost-Effective I/O 型单片机具有良好运行性能的主要因素。由于采用 RISC 结构，此系列单片机具有高运算速度和高性能的特性。采用流水线的方式，即指令的取得和执行同时进行，使得除了分支、调用和查表指令外，其他指令都能在一个指令周期内完成。8 位的 ALU 参与指令集中所有的运算。它可完成算术运算、逻辑运算、移位、加、减和分支等功能。内部的数据路径则以通过累加器或 ALU 的方式加以简化。有些寄存器在数据存储器中被实现，且可以直接或间接寻址。简单的寄存器寻址方式和结构特性，确保了提供最大可靠度和灵活性的输入/输出控制系统时，仅需要少数的外部器件。这使该系列单片机适合用在需要 0.5k 至 1k 字的程序存储器和 32 至 64 字节数据储存器的控制应用上。

时序和流水线结构(Pipelining)

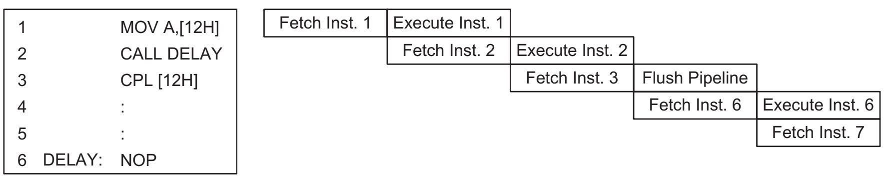
系统时钟由晶体/陶瓷振荡器，或是由 RC 振荡器提供，细分为 T1~T4 四个内部产生的非重叠时钟。程序计数器在 T1 时自动加一并抓取一条新的指令。剩下的 T2~T4 时钟完成解码和执行。因此一个 T1~T4 时钟组成一个指令周期。虽然指令的取得和执行发生在连续的指令周期，但单片机流水线的结构会保证指令在一个指令周期内被有效的执行。特殊的情况发生在程序计数器的内容被改变的情形，如子程序的调用或跳转，在这情况下指令将需要多一个指令周期的时间去执行。

注意：当使用 RC 振荡器时，OSC2 可以如同一个 T1 相时钟同步引脚一样地被使用，这个 T1 相时钟有 $f_{SYS}/4$ 的频率，占空比为 1:3。



系统时序和流水线

如果指令牵涉到分支，例如跳转或调用等指令，则需要两个指令周期才能完成指令执行。需要一个额外周期的原因是程序先用一个周期取出当前指令地址的下一条指令，再用另一个周期去实际执行分支动作，因此编程者必须特别考虑额外周期的问题，尤其是在执行时间要求比较严格的时候。



程序计数器

程序执行期间，程序计数器是用来指向下一条要执行的指令地址。除了 JMP 或 CALL 这些要求跳转到一个非连续的程序存储器地址外，它会在每条指令执行完后自动增加一。对于低成本、高效益的输入/输出系列，HT48R05A-1 和 HT48R06A-1 程序计数器分别有 9 和 10 个位。然而需要注意只有较低的 8 位，即所谓的程序计数器低字节寄存器，是可以直接读写的。

当执行的指令要求跳转到非连续的地址时，如跳转指令、子程序调用、中断或复位等，单片机通过载入所需的地址到程序计数器来控制程序。对于条件跳转指令，一旦条件符合，下一条在现在指令执行时所取得的指令即会被摒弃，而由一个空指令周期来加以取代。

程序计数器较低字节，即程序计数器低字节寄存器或 PCL，可以通过程序控制取得，且它是可以读取和写入的寄存器。通过直接传送数据到这寄存器，一个程序短跳转可以直接被执行，然而因为只有低字节的运用是有效的，因此跳转被限制在同页存储器，即 256 个存储器地址的范围内。当这样一个程序跳转要执行时，需注意会插入一个空指令周期。

注意：程序计数器较低字节在程序控制下是可以访问的。PCL 的使用可能导致程序分支，所以额外的周期需要预先取得。有关 PCL 寄存器更多的信息可在特殊功能寄存器部份中找到。

模式	程序计数器									
	b9	**b8**	**b7**	**B6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
复位	0	0	0	0	0	0	0	0	0	0
外部中断	0	0	0	0	0	0	0	1	0	0
定时/计数器中断	0	0	0	0	0	0	1	0	0	0
条件跳转	Program Counter + 2									
写入 PCL 寄存器	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
跳转或调用子程序	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
由子程序返回	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

注意：1.PC9~PC8：目前程序计数器位

2.@7~@0：PCL 位

3.#9~#0：指令码位

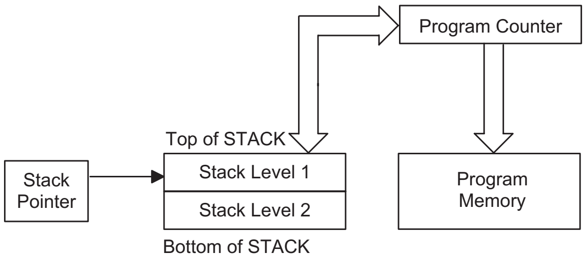
4.S9~S0：堆栈寄存器位

5.对于 HT48R06A-1，程序计数器有 10 个位，即从 b9~b0，对于 HT48R05A-1，它的程序计数器有 9 个位，表中的 b9 是无效的。

堆栈

堆栈是存储器的一个特殊部分，只用来储存程序计数器中的内容。堆栈由两层组成，既不是数据部分也不是程序空间部份，且既不是可读取也不是可写入的。当前层由堆栈指针(Stack Pointer, SP)加以指示，同样也是不可读写的。在发生子程序调用或中断响应服务时，程序计数器的内容被压入堆栈。当子程序或中断服务程序结束时，返回指令(RET 或 RETI)使程序计数器从堆栈中重新得到它以前的值。当一个芯片复位之后，SP 将指向堆栈的顶部。

如果堆栈已满，且有非屏蔽的中断发生，中断请求标志位会被置位，但是中断响应会被禁止。当堆栈指针减少(执行 RET 或 RETI)，中断将被响应。这个特性提供程序设计者简单的方法来预防堆栈溢出。然而即使堆栈已满，CALL 指令仍然被执行，而造成堆栈溢出。使用时应注意避免堆栈溢出情况的发生，因为这可能会造成不可预期的程序分支指令执行的错误。最先被存入的地址会丢失，只有最后两个返回地址被保存在堆栈里。



算术及逻辑单元 – ALU

算术逻辑单元是单片机中很重要的部份，执行指令集中的算术和逻辑运算。ALU 连接到单片机的数据总线，在接收相关的指令码后执行需要的算术与逻辑操作，并将结果储存在指定的寄存器。当 ALU 计算或操作时，可能导致进位、借位或其他状态的改变，而相关的状态寄存器会因此更新内容以显示这些变化，ALU 所提供的功能如下：

- 算术操作：ADD、ADDM、ADC、ADCM、SUB、SUBM、SBC、SBCM、DAA
- 逻辑运算：AND、OR、XOR、ANDM、ORM、XORM、CPL、CPLA
- 移位：RRA、RR、RRCA、RRC、RLA、RL、RLCA、RLC
- 增加和减少：INCA、INC、DECA、DEC
- 分支判断：JMP、SZ、SZA、SNZ、SIZ、SDZ、SIZA、SDZA、CALL、RET、RETI

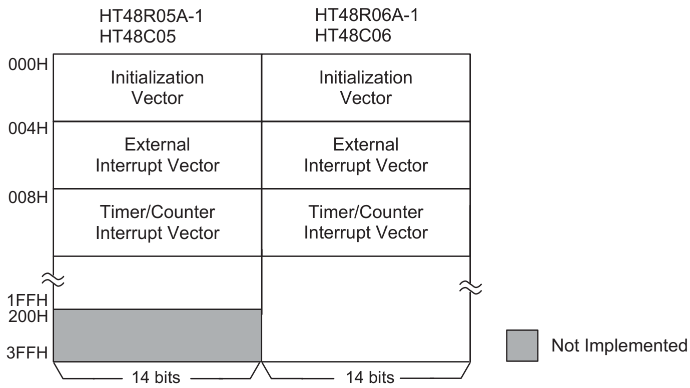
程序存储器

程序存储器用来存放用户代码即存储程序。对于 Cost-Effective I/O 型的单片机而言，有两种程序存储器可供使用。第一种是一次可编程存储器(OTP)，使用者可编写他们的应用码到芯片中.具有 OTP 存储器的单片机在芯片名称上有“R”做标示。使用适当的编程工具，OTP 单片机可以提供使用者以灵活的方式来自由开发他们的应用，这对于除错或需要经常升级与改变程序的产品是很有帮助的。对于中小型量产，OTP 亦为极佳的选择。另一种存储器为掩膜存储器，单片机名称上有“C”做标示。这些芯片对于大量生产提供最佳的成本效益。

结构

14 位的程序存储器的容量是 0.5K 或 1K，是使用者用来储存程序的地方。程序存储器用程序计数器来寻址，其中包含数据、表格和中断入口。数据表格可以设定在程序存储器的任何地址，由表格指针来寻址。

以下是 Cost-Effective I/O 型单片机程序存储器结构图。



特殊向量

程序存储器内部某些地址保留用做诸如复位和中断入口等特殊用途。

- 地址 000H

  这个向量是芯片复位后的程序起始地址。在芯片复位之后，程序将跳到这个地址并开始执行。

- 地址 004H

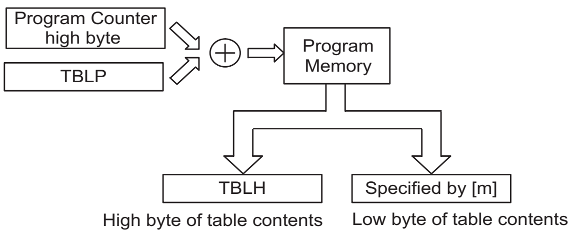
  这个向量用做外部中断入口。假如单片机外部中断引脚电平转成低电平，而外部中断使能且堆栈没有满的情况下，程序将跳到这个地址开始执行。

- 地址 008H
  此内部中断向量被定时/计数器所使用。当定时器发生溢出，而内部中断使能且堆栈没有满的情况下，程序将跳到这个地址开始执行。

查表

程序存储器中的任何地址都可以定义成一个表格，以便让编程者可以储存固定的数据。使用表格时，表格指针必须先行设定，其方式是将表格的低字节地址放在表格指针寄存器 TBLP 中。这个寄存器存放表格较低的 8 位地址。在设定完表格指针后，表格数据可以使用“TABRDC [m]”指令从当前程序所在的存储器页中来查表读取。在单片机 HT48R06A-1/HT48C06 中，另一条“TABRDL [m]”指令可用来读取程序存储器最后页中的表格数据，但这指令对于单片机 HT48R05A-1/HT48C05 并不适用。当这些指令执行时，程序存储器中表格数据的低字节，将被传送到使用者指定的数据存储器。程序存储器中表格数据的高字节，则被传送到 TBLH 特殊寄存器。而高字节中未使用的位将读取为 0。

下图是查表中寻址/数据流程图：



查表程序范例

以下范例说明 HT48R06A-1 输入/输出型单片机中，表格指针和表格数据如何被定义，如何读取表格数据。这个例子使用的表格数据用 ORG 伪指令储存在存储器最后一页。 在此 ORG 伪指令中的值为 300H，即 1K 程序存储器 HT48R06A-1 单片机中存储器最后一页的开始地址。这里设定表格指针 TBLP 的初始值为 06H 。这说明从数据表格读取的第一笔数据位于程序存储器地址 306H 即最后一页开始地址后的第六个地址。值得注意的是假如执行“TABRDC [m]”指令，则表格指针指向当前页。在这个例子中，表格数据的高字节等于零，当“TABRDL [m]”指令被执行时，此值将会自动的传送到 TBLH 寄存器。

```
tempreg1   db     ? ;temporary register #1
tempreg2   db     ? ;temporary register #2
      :
      :

mov        a,06h    ;initialize table pointer - note that this
                    ;address is referenced

mov        tblp,a   ;to the last page or present page
      :
      :

tabrdl   tempreg1   ;transfers value in table referenced by table
                    ;pointer to tempreg1
                    ;data at prog.memory address 306H transferred to
                    ;tempreg1 and TBLH

dec        tblp     ;reduce value of table pointer by one

tabrdl   tempreg2   ;transfers value in table referenced by table
                    ;pointer to tempreg2
                    ;data at prog.memory address 305H transferred to
                    ;tempreg2 and TBLH
                    ;in this example the data ˝1A˝ is transferred
                    ;to tempreg1 and data ˝0F˝ to register tempreg2
                    ;the value ˝0˝ will be transferred to the high
                    ;byte register TBLH
      :
      :

org  300h           ;sets initial address of last page
                    ;(for HT48R06A-1)

dc   00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
      :
      :
```

因为 TBLH 寄存器是只读寄存器不能写入，若主程序和中断服务程序都使用表格读取指令，应该注意对它的保护。使用表格读取指令，中断服务程序可能会改变 TBLH 的值，若随后在主程序中再次使用这个值，则会发生错误。因此建议避免同时使用表格读取指令。然而在某些情况下，如果同时使用表格读取指令是不可避免的，则在执行任何主程序的表格读取指令前，中断应该先除能。另外要注意的是所有与表格相关的指令，都需要两个指令周期去完成操作。

指令	表格地址									
	b9	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
TABRDC[m]	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL[m]	1	1	@7	@6	@5	@4	@3	@2	@1	@0

注意： 1. PC9 ~ PC8：当前程序计数器位

2. @7 ~ @0：表格指针 TBLP 位

3. 对 HT48R05A-1/HT48C05 来说，表格地址是 9 位，从 b8~b0。

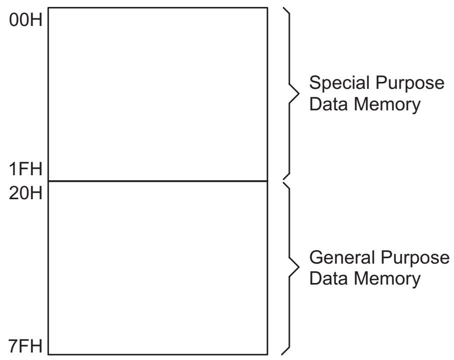
4. 对 HT48R06A-1/HT48C06 来说，表格地址是 10 位，从 b9~b0。

数据存储器

数据存储器是内容可更改的 8 位 RAM 内部存储器，用来储存临时数据。它分为两部份，第一部份是特殊功能寄存器。这些寄存器有固定的地址且与单片机的正确操作密切相关。大多特殊功能寄存器都可在程序控制下直接读取和写入，但有些被加以保护而不对用户开放。第二部份数据存储器是做一般用途使用，都可在程序控制下进行读取和写入。

结构

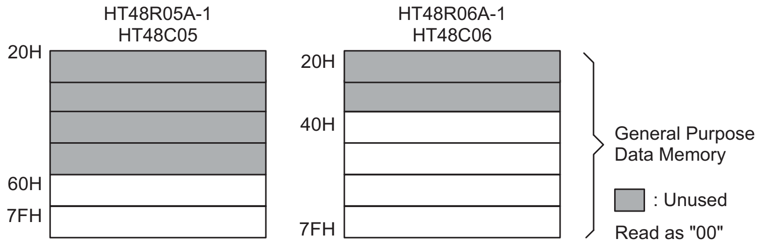
数据存储器的两个部份，即专用和通用数据存储器，位于连续的地址。全部 RAM 为 8 位宽度，但存储器容量大小因所选择的单片机而不同。所有芯片的数据存储器开始地址都是 00H。单片机常见的寄存器，如 ACC 和 PCL 等，全都位于相同的数据存储地址。



注意： 除了少数专用的位，大部份数据存储器的位都可以直接使用“SET [m].i”和“CLR [m].i”加以操作。数据存储器也可通过间接寻址指针寄存器(MP)进行存取。

通用数据存储器

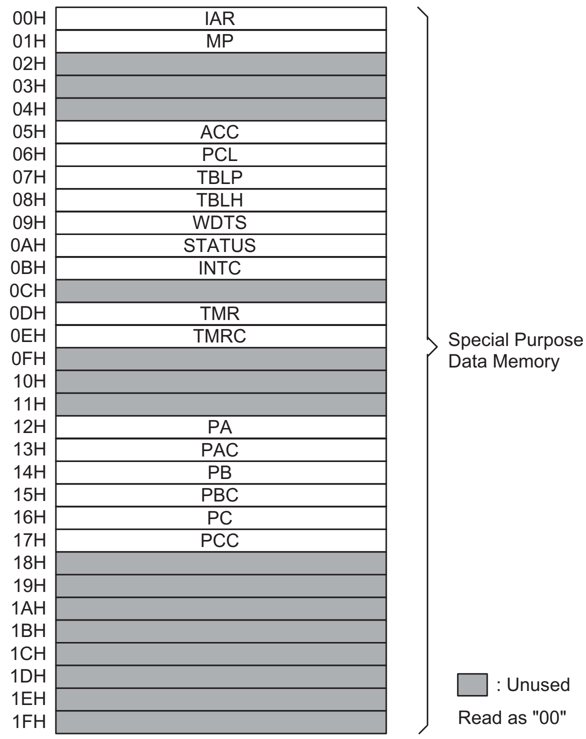
所有的单片机程序需要一个读/写的存储区，让临时数据可以被储存和再使用。该 RAM 区域就是通用数据存储器。这个数据存储区可让使用者进行读取和写入的操作。使用“SET [m].i”和“CLR [m].i”指令可对个别的位做置位或复位的操作，方便用户在数据存储器内进行位操作。



专用数据存储器

这个区域的数据存储器是存放特殊寄存器的，这些寄存器对单片机的正确操作密切相关。大多数的寄存器可被读取和写入，只有一些是被保护而只能读取的。相关细节的介绍请参看有关特殊功能寄存器的部份。要注意的是，任何读取指令对存储器中未使用的地址进行读取将得到 00H 的值。

以下是 Cost-Effective I/O 型单片机中，专用数据存储器的详细结构图。



特殊功能寄存器

为了确保单片机能成功的操作，数据存储器中设置了一些内部寄存器。这些寄存器确保内部功能，如定时器、中断和看门狗等，和外部功能，如输入/输出数据控制的正确操作。在数据存储器中，这些寄存器以 00H 作为开始地址。在特殊功能寄存器存储空间和通用数据存储器的起始地址之间，有一些未定义的数据存储器，将被保留用来做未来的扩充，若从这些地址读取数据将返回 00H 值。

间接寻址寄存器 – IAR

IAR 寄存器位于数据存储器 00H 地址，没有实际的物理空间。这个特殊寄存器提供间接寻址，准许使用间接寻址指针做数据操作，以取代定义实际存储器地址的直接存储器寻址方式。对 IAR 上的任何动作，将对间接寻址指针(MP)所指定的存储器地址产生对应的读/写操作。直接读取 IAR 寄存器将返回 00H 的结果，而直接写入此寄存器则不做任何操作。

间接寻址指针 – MP

间接寻址指针(MP)位于数据存储器的物理空间内。它像普通的寄存器一样被写入和操作，这提供了寻址和数据追踪的有效方法。对 IAR 操作时，单片机将对 MP 指向的地址操作。

注意： 间接寻址指针的第七位没有用处。然而要注意的是，当系统去读间接寻址指针的内容时，此位为“1”。

以下的例子说明如何清零一个具有 4 个 RAM 地址的区块，它们已事先被定义成地址 adres1 到 adres4。

```
data .section ´data´
adres1     db ?
adres2     db ?
adres3     db ?
adres4     db ?
block      db ?
code .section at 0 ´code´
org   00h
start:
   mov a,04h            ;setup size of block
   mov block,a
   mov a,offset adres1  ;Accumulator loaded with first RAM address
   mov mp,a             ;setup memory pointer with first RAM address
loop:
   clr [00h]            ;clear the data at address defined by mp
   inc mp               ;increment memory pointer
   sdz block            ;check if last memory location has been
                        ;cleared
   jmp loop
continue:
```

在上面的例子中有一点值得注意，即并没有确定 RAM 地址。

累加器 – ACC

对任何单片机来说，累加器是相当重要的且与 ALU 所完成的运算有密切关系。所有 ALU 得到的运算结果都会暂时储存在 ACC 累加器里。若没有累加器，ALU 必须在每次进行如加法、减法和移位的运算时，将结果写入到数据存储器，这样将造成程序编写和时间的负担。另外数据传送也常常牵涉到累加器的临时储存功能，例如在一个使用者定义的寄存器和另一个寄存器之间传送数据时，由于两寄存器之间不能直接传送数据，因此必须通过累加器来传送数据。

程序计数器低字节寄存器 – PCL

为了提供额外的程序控制功能，程序计数器低字节设置在数据存储器的特殊功能区域内，程序员可对此寄存器进行操作，很容易的直接跳转到其它程序地址。直接给 PCL 寄存器赋值将导致程序直接跳转到程序存储器的某一地址，然而由于寄存器只有 8 位的长度，因此只允许在本页的程序存储器范围内进行跳转。而当使用这种运算时，要注意会插入一个空指令周期。

表格寄存器 – TBLP, TBLH

这两个特殊功能寄存器用来对储存在程序存储器中的表格进行操作。TBLP 为表格指针，指向表格的地址。它的值必须在表格读取指令执行前加以设定。由于它的值可以被如 INC 或 DEC 的指令所改变，这就提供了一种简单的方法对表格数据进行读取。表格读取数据指令执行之后，表格数据高字节存储在 TBLH 中。其中要注意的是，表格数据低字节会被传送到使用者指定寄存器。

看门狗定时寄存器 – WDTS

单片机中的看门狗特性提供自动复位功能，给予单片机一个保护工具去预防不正确的程序跳转。当看门狗定时器溢出时会产生复位。为了提供可变的看门狗定时器复位时间，看门狗定时器的时钟源可被预分频，分频值由 WDTS 寄存器来设定。对 WDTS 寄存器赋值，可以设定适当的预分频值的看门狗定时器时钟源。要注意的是，在 WDTS 中低 3 位被使用来设定从 1 到 128 之间的分频比例，8 位中剩下的 5 位可以被程序设计者用来做其它用途。

状态寄存器 – STATUS

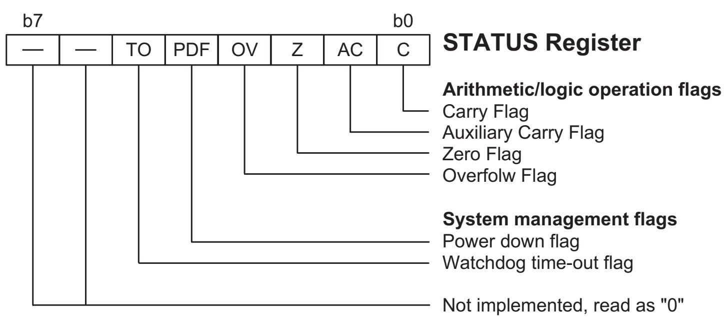
这 8 位寄存器(0AH)包含零标志位(Z)、进位标志位(C)、辅助进位标志位(AC)、溢出标志位(OV)、暂停标志位(PDF)和看门狗溢出标志位(TO)。它同时记录状态数据和控制运算顺序。

除了 TO 和 PDF 标志位外，状态寄存器中的位像其它大部份寄存器一样可以被改变。但任何数据写入到状态寄存器都不会改变 TO 或 PDF 标志位。另外，执行不同的指令后，与状态寄存器有关的运算可能会得到不同的结果。TO 标志位只会受系统上电、看门狗溢出、或执行“CLR WDT”或“HALT”指令影响。PDF 标志位只会受执行“HALT”或“CLR WDT”指令或系统上电影响。

Z、OV、AC 和 C 标志位通常反映最近运算的状态：

- 当加法运算的结果产生进位，或减法运算的结果没有产生借位时，置位 C；反之 C 被　　清零。同时 C 也会被带进位/借位的移位指令所影响。
- 当低半字节加法运算的结果产生进位，或高半字节减法运算的结果没有产生借位时，置位 AC；反之 AC 被清零。
- 当算术或逻辑运算结果是零时，置位 Z；反之 Z 被清零。
- 当运算结果高两位的进位状态异或的结果为 1 时，置位 OV；反之 OV 被清零。
- 系统上电或执行“CLR WDT”指令会清零 PDF，而执行“HALT”指令则会置位 PDF。
- 系统上电或执行“CLR WDT”或“HALT”指令会清零 TO，而当 WDT 溢出则会置位 TO。

另外当进入一个中断程序或执行子程序调用时，状态寄存器不会自动压入到堆栈保存。假如状态寄存器的内容是重要的且子程序可能改变状态寄存器的话，则需谨慎的去做正确的储存。



中断控制寄存器 – INTC

8 位的 INTC 寄存器控制外部和内部中断的动作。通过使用标准的位操作指令来设定这寄存器的位的值，外部中断和内部中断的使能和除能功能可分别被控制。寄存器中的主中断位(EMI)控制所有中断的使能/除能，用来设定所有中断使能位的开或关。当一中断程序被响应时，就会自动屏蔽其他中断，EMI 位将被清零，而执行“RETI”指令则会置位 EMI 位。

注意： 如果在当前中断服务程序中要再响应其它的中断程序，可以在进入该中断服务程序后，在程序中用手动的方式将 EMI 置“1”。

定时/计数器寄存器

所有单片机都具有一个内部 8 位向上定时/计数器，即所谓的 TMR，而它的内部时钟源则来自系统时钟 f_{SYS}。PC1/TMR 输入引脚用来使定时器能计算外部事件、测量时间间距和脉冲宽度，定时器同时也可以用来产生精确的时基或 PFD 信号。系统提供 TMR 和 TMRC 两个特殊功能寄存器，以控制定时器的动作。TMR 寄存器储存实际定时器计数值，而 TMRC 寄存器则用来设定定时器的各种功能选择项。TMR 寄存器可以预先载入固定的数据，以允许不同的时间中断。定时器将从这预先载入的值开始计数直到溢出为止，随后即产生一个内部中断信号，并且 TMR 寄存器会自动重新载入此预先载入值。

输入/输出端口和控制寄存器

在特殊功能寄存器中，输入/输出寄存器和它们相对应的控制寄存器是很重要的。所有的输入/输出端口都有相对应的寄存器，且被标示为 PA 到 PC。如数据存储器结构图中所示，这些输入/输出端口寄存器映射到数据存储器的特定地址，用以传送端口上的输入/输出数据。每个输入/输出端口有一个相对应的控制寄存器，分别为 PAC、PBC 和 PCC，也同样映射到数据存储器的特定地址。这些控制寄存器设定引脚的状态，以决定哪些是输入口，哪些是输出口。要设定一个引脚为输入，控制寄存器对应的位必须设定成高，若引脚设定为输出，则控制寄存器对应的位必须设为低。程序初始化期间，在从输入/输出口中读取或写入数据之前，必须先设定控制寄存器的位以确定引脚为输入或输出。使用“SET [m].i”和“CLR [m].i”指令可以直接设定这些寄存器的某一位。这种在程序中可以通过改变输入/输出端口控制寄存器中某一位而直接改变该端口输入/输出口状态的能力是此系列单片机非常有用的特性。

输入/输出端口

盛群单片机的输入/输出端口控制具有很大的灵活性。这体现在每一个引脚在使用者的程序控制下可以被指定为输入或输出、所有引脚的上拉选项、以及特定引脚的唤醒选择，这些特性使此类单片机在广泛应用上都能符合开发的要求。

依封装类别区分，该系列单片机最多有 13 个双向输入/输出口，标示为 PA、PB 及 PC，而 PA、PB 及 PC 对应的数据存储器则分别为[12H]、[14H]及[16H]。所有输入/输出端口都可做为输入及输出之用。作为输入操作时，输入/输出引脚是不锁存的，也就是输入数据必须在指令“MOV A, [m]”（m=12H、14H 或 16H）的 T2 上升沿准备好。对于输出操作，所有数据是锁存的，而且持续到输出锁存被重写。

上拉电阻

很多产品应用在端口处于输入状态时需要外加一个上拉电阻来实现上拉的功能。为了免去这个外加的电阻，当引脚规划为输入时，可由内部连接到一上拉电阻，这些上拉电阻可通过掩膜选项来加以选择，它用一个 PMOS 晶体管来实现。要注意的是一旦选择了上拉电阻选项，则所有引脚都将被连接到上拉电阻。

PA 口的唤醒

本系列的单片机都具有暂停功能，使得单片机进入暂停模式以节省功耗，此功能对于电池及低功率应用是很重要的。唤醒单片机有很多种方法，其中之一就是使 PA 端口其中的一个引脚从高电平转为低电平。当使用暂停指令迫使单片机进入暂停状态以后，单片机将保持闲置即低功率状态，直到 PA 口上被选为唤醒输入的引脚电平发生下降沿跳变。这个功能特别适合于通过外部开关来唤醒的应用。值得注意的是 PA 端口的每个引脚可单独的选择具有唤醒功能。

输入/输出端口控制寄存器

每一个输入/输出口都拥有自己的控制寄存器（PAC、PBC、PCC）去控制输入/输出状态。利用此控制寄存器，每一个 CMOS 输出或者 Schmitt 触发器输入可利用软件控制方式加以动态的重新设置。所有输入/输出端口的引脚都各自对应于输入/输出端口控制寄存器的某一位。若要实现输入功能，则对应的控制寄存器位必须设定为“1”。这时程序指令可以直接读出输入引脚的逻辑状态。如果引脚的控制寄存器位被设定为“0”，则此引脚被设置为 CMOS 输出。当引脚被设置为输出状态，程序指令读取的是输出端口寄存器的内容。请注意当输入/输出口被设置为输出状态时，此时如果对输出口做读取的动作，则会读取到内部数据寄存器中的锁存值，而不是输出引脚实际的逻辑状态。

引脚共用功能

如果引脚能有超过一个以上的功能，则单片机灵活程度将大大的提升。有限的引脚个数会严重地限制设计者，但是引脚的多功能特性，可以解决很多此类问题。多功能输入/输出引脚的功能选择，有些是由掩膜选项设定，另一些则是在应用程序控制时做设定。

→ 蜂鸣器

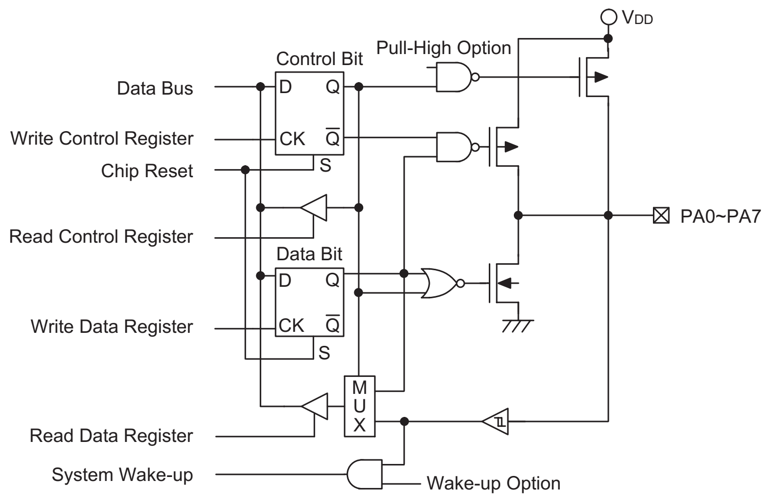
蜂鸣器引脚 BZ 及 $\overline{BZ}$ 与输入/输出引脚 PB0 及 PB1 所共用。假如定义为蜂鸣器引脚，则需选择正确的硬件及软件选项。

→ 外部中断输入

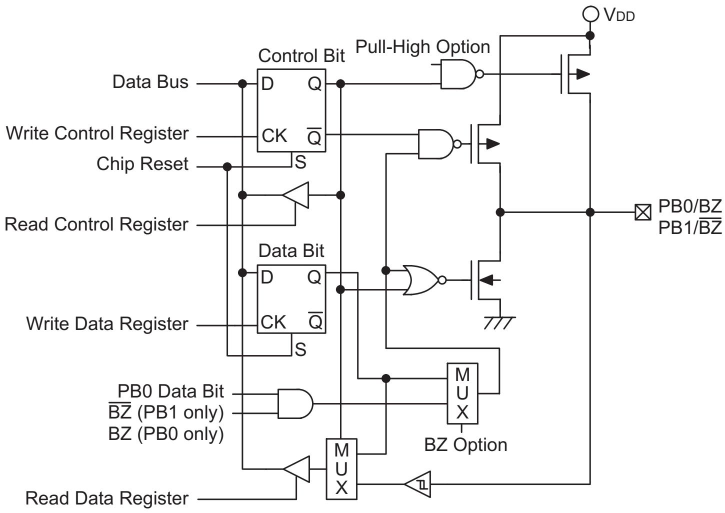
外部中断引脚 $\overline{INT}$ 与输入/输出引脚 PC0 共用。要把这个引脚做为外部中断引脚，而不是一般输入/输出引脚，必须正确地设定中断控制寄存器 INTC 里的外部中断允许位。如果引脚不作外部中断输入，那么可以作一般的输入/输出引脚使用，此时 INTC 寄存器中的外部中断使能位必须被关断

→ 外部定时器时钟输入

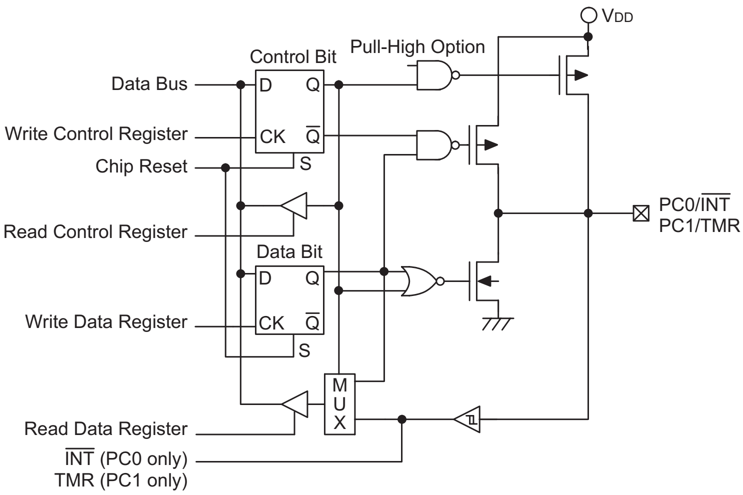
外部定时器引脚 TMR 与输入/输出引脚 PC1 共用。为了把这个引脚做为定时器输入，必须正确地设置 TMRC 定时器寄存器的控制位。在不需要外部定时器输入引脚的场合，此引脚也可以当作一般输入/输出引脚使用。要注意的是在做为一般输入/输出引脚使用时，TMRC 寄存器中的定时器模式位必须选为定时器模式(内部时钟来源)，以避免输入/输出引脚与定时器操作的冲突。



PA 输入/输出端口



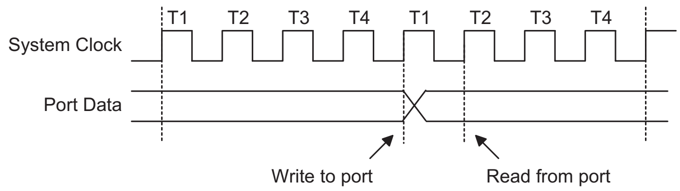
PB0/PB1 输入/输出端口



PC0/PC1 输入/输出端口

编程注意事项

在使用者的程序中，最先要考虑的是端口的初始化。复位之后，所有的输入/输出数据及端口控制寄存器都将被设为逻辑高。意思是说所有输入/输出引脚将被默认为输入状态，而引脚上的电平则取决于外部连接电路以及是否选择了上拉选项。假如 PAC、PBC、PCC 等端口控制寄存器接着设定某些引脚为输出之用，则这些输出引脚会有初始高输出值，除非相关端口数据寄存器 PA、PB、PC 被预先设定。要选择哪些引脚是输入及哪些引脚是输出，可通过载入正确的值到适当的端口控制寄存器，或者使用指令“SET [m].i”及“CLR [m].i”来设定端口控制寄存器中个别的位。要注意的是当使用这些位控制指令时，一个读-修改-写的操作将会发生。单片机必须先读入整个端口上的数据，修改个别位的值，然后重新把这些数据写入到输出端口。



PA 口有唤醒的额外功能。当芯片在 HALT 状态时有很多方法去唤醒此单片机，其中之一就是端口 PA 任一个引脚电平由高到低的转换。可以设定 PA 口的一个或多个引脚有这项功能。

定时/计数器

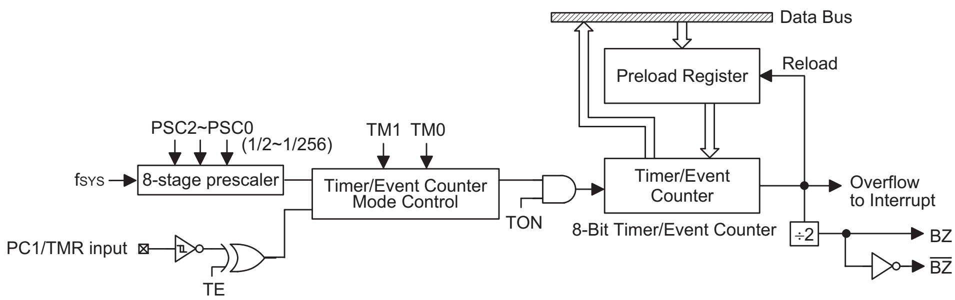
定时器在任何单片机都是一个很重要的部分，也提供程序设计者一种实现和时间有关功能的功能。每一个芯片内部包含一个 8 位向上定时计数器，定时器包含三种操作模式，可被当作一个普通定时器、外部的事件计数器、或者脉冲宽度测量器使用。8 级预分频器(Prescaler)也加大了定时器的定时范围。

两个和定时/计数器相关的寄存器为 TMR[0DH]及 TMRC[0EH]。TMR 寄存器中存放实际的计时值。写入 TMR 会将一个初始值放在定时/计数器的预载寄存器中，而读取 TMR 则会重新得到定时/计数器的内容。TMRC 是定时/计数器的控制寄存器，它设置定时/计数器的选项，控制定时/计数器的使用。定时器的时钟来源可来自内部时钟源或是共用引脚 PCI/TMR 上的外部时序。

配置定时/计数器输入时钟源

内部定时器的计时来源可以来自系统或外部计时来源。当定时器在定时器模式或者在脉冲宽度测量模式时，使用系统时钟作为计时来源。内部定时器时钟来源也经过一个预分频器(Prescaler)，预分频值由 PSC0、PSC1 及 PSC2 三位决定。

定时器在事件计数模式时使用外部计时来源，计时来源是由 PCI/TMR 共同引脚来提供。当 PCI/TMR 引脚由高电平到低电平或者由低电平到高电平（由 TE 位决定）进行转换时，定时器值加一。



8 位定时/计数器结构

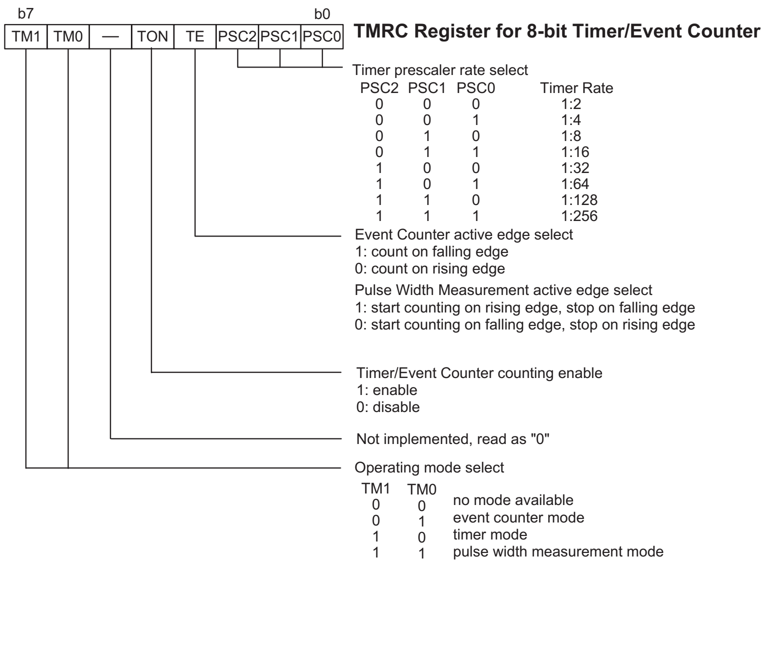
定时/计数寄存器 – TMR

TMR 寄存器是位于专用数据存储器内 8 位的特殊功能寄存器，具有储存实际定时器值的用途。在用作内部定时且收到一内部计时脉冲时，或用作外部计数且引脚 PC1/TMR 发生一外部状态跳变时，此寄存器的值将会加一。定时器将从预置寄存器所载入的值开始计数，直到 FFH 储满，此时定时器溢出且会产生一个内部中断信号。定时器的值随后被预置寄存器的值重设并继续计数。为了得到从 00H 到 FFH 的最大计算范围，预置寄存器必须先清除为 00H。此时要注意的是上电后预置寄存器处于未知状态。定时/计数器在 OFF 条件下，如果把数据写入预置寄存器，这数据将被立即写入实际的定时器。然而如果定时/计数器已经被打开且正在计数，在这个周期内写入到预置寄存器的任何新数据将保留在预置寄存器，只有在下一个溢出发生时才被写入实际定时器。当 TMR 寄存器被读取时，定时器的计时时钟会停止计数以避免错误，然而这可能造成某些时序的错误，因此程序设计者必须考虑到这点。

定时/计数控制寄存器 – TMRC

定时/计数器能工作在三种不同的模式，至于选择工作在哪一种模式则是由定时/计数控制寄存器 TMRC 的内容决定。TMRC 和 TMR 两个寄存器控制定时/计数器的全部操作，在定时器使用之前必须先正确地设定 TMRC 寄存器，以便保证定时器能正确操作，而这个过程通常在程序初始化期间完成。

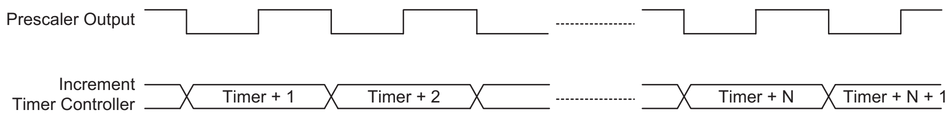
为了确定定时器工作在哪一种模式，位 TM0 和 TM1 必须设定到要求的逻辑电平。定时器打开位 TON，即定时/计数控制寄存器的第 4 位，是定时器控制的开关，设定逻辑高时定时器开始计数，而清零时则定时器停止计数



TMRC 寄存器的位 0~2 决定输入定时预分频器(Prescaler)中的分频比例。如果使用外部计时源，预分频器(Prescaler)的位将不作用。如果定时器工作在事件计数或脉冲宽度测量模式，TE 的逻辑电平即 TMRC 寄存器的第 3 位将可用来选择上升沿或下降沿触发。

定时器模式

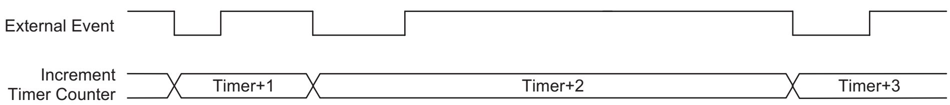
在这个模式，定时器可以用来测量固定时间间距，当定时器发生溢出时，就会提供一个内部中断信号。要工作在这个模式，TMRC 寄存器中位 TM1(bit7)和 TM0(bit6)必须分别设为 1 和 0。在这个模式，内部时钟源被用来当定时器的计时源。定时器的输入计时频率是 f_{SYS} 除以定时器预分频器(Prescaler)的值，这个值是由 TMRC 寄存器的 PSC0~PSC2 位来决定。定时器打开位 TON 必须被设为逻辑高，才能令定时器工作。每次内部时钟由高到低的电平转换都会使定时器值增加一；当定时器已满即溢出时，会产生中断信号且定时器会重新载入已经载入到预置寄存器的值，然后继续计数。定时器溢出是中断的一种，也是唤醒暂停模式的一种方法。



定时器模式时序图

事件计数器模式

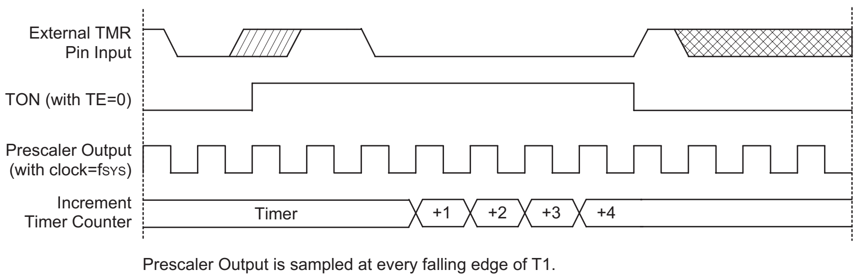
在这个模式，发生在外部引脚 PC1/TMR 的外部逻辑事件改变的数目，可以通过内部定时/计数器来记录。为使定时/计数器工作在事件计数模式，TMRC 寄存器中位 TM1 和 TM0 必须分别设为 0 和 1。计数器打开位 TON 必须设为逻辑高，令计数器开始计数。当 TE 为逻辑低时，每次 PC1/TMR 引脚接收到由低到高电平的转换将使计数器加一。而当 TE 为逻辑高时，每次 PC1/TMR 引脚接收到由高到低电平的转换将使计数器加一。与另外两个模式一样，当计数器计满时，计数器将溢出且产生一个内部中断信号，同时计数器将重新载入已经载入到预置寄存器的值。为了确保 TMR/PC1 引脚工作在事件计数器模式，要注意两点。第一点是要将 TM0 与 TM1 位设定在事件计数器模式，第二点是确定此引脚的输入/输出口控制寄存器对应位被设定为输入状态。计数器溢出是中断的一种，也是唤醒暂停模式的一种方法。



事件计数器模式时序图

脉冲宽度测量模式

在这个模式，可以被测量外部引脚 PC1/TMR 的外部脉冲宽度。在脉冲宽度测量模式中，定时/计数器时钟源是由内部时钟提供。位 TM0 和 TM1 则必须都设为逻辑高。如果 TE 位是逻辑低，当 PC1/TMR 引脚接收到一个由高到低电平的转换时，定时/计数器将开始计数直到 PC1/TMR 引脚回到它原来的高电平。此时 TON 位将自动清除为零，且定时/计数器停止计数。而如果 TE 位是逻辑高，则当 PC1/TMR 引脚接收到一个由低到高电平的转换时，定时/计数器开始计数直到 PC1/TMR 引脚回到原来的低电平。如上所述，TON 位将自动清除为零，且定时/计数器停止计数。请注意，在脉冲宽度测量模式中， TON 位将自动地清除为零且定时/计数器会停止计数，而在其他两种模式下，TON 位要在程序控制下才会被清除为零。这时定时/计数器中的剩下的值被程序读取，可以由此得知 PC1/TMR 引脚接收到的脉冲的长度。当 TON 位被清零时，任何在 PC1/TMR 引脚的进一步的电平转换将被忽略。直到 TON 位再次被程序设定为逻辑高，定时/计数器才又开始脉冲宽度测量。利用这种方法可以轻松完成单一脉冲测量。要注意的是在这种模式下，定时/计数器是通过 PC1/TMR 引脚上的逻辑转换来控制，而不是通过逻辑电平。与另外两个模式一样，当定时/计数器已满会发生溢出，且产生一个内部中断信号，定时/计数器也将清零并载入预置寄存器的值。为了确保 TMR/PC1 引脚工作脉冲宽度测量模式，要注意两点。第一点是要将 TM0 与 TM1 位设定在脉冲宽度测量模式，第二点是确定此引脚的输入/输出口控制寄存器对应位被设定为输入状态。这种定时器溢出是中断的一种，也是唤醒暂停模式的一种方法。



脉冲宽度测量模式时序图

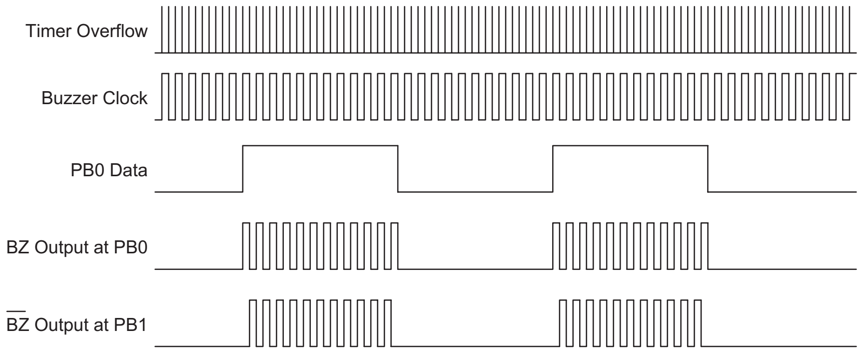
可编程分频器(PFD)和蜂鸣器的应用

和可编程分频器的作用相似，单片机中蜂鸣器的功能在于能提供可变频率输出，以适用于象压电蜂鸣器驱动、或其它需要精确频率输出的场合。

该系列所有的单片机根据封装不同，可提供一个 BZ 或一对 BZ/$\overline{BZ}$ 输出，这个功能是通过掩膜选项选择的。如果不选择为蜂鸣器输出，其功能即如正常的输入/输出引脚。要注意的是$\overline{BZ}$引脚是 BZ 引脚的反向输出，两者配合可以产生更强的输出功率驱动蜂鸣器等器件。

蜂鸣器电路使用定时器溢出信号作为其时钟源。载入合适的值到定时器预分频器(Prescaler)，可以产生需要的时钟源分频比例，由此来控制输出的频率。系统时钟被预分频器(Prescaler)分频后的时钟源，进入定时器计时，定时器从预置寄存器的值开始往上计算，直到计数值满并产生溢出信号，并改变 BZ/$\overline{BZ}$ 输出状态。定时器将自动地重新载入预置寄存器的值，并继续向上计数。具体设置和运作细节可参考定时/计数器章节。

要蜂鸣器正确工作，必须将端口 PB 控制寄存器 PBC 的第 0 位和第 1 位设置为输出。若它们设置为输入，则蜂鸣器输出将不会动作，对应引脚作为正常输入/输出口。只有把 PB0 位置“1”，BZ/$\overline{BZ}$ 输出引脚才会有输出。这个输出数据位被用来作为 BZ/$\overline{BZ}$ 输出的开关控制。注意，PB0 为“0”的话，BZ/$\overline{BZ}$ 输出都为低电平。注意，PB1 对 BZ/$\overline{BZ}$ 输出控制不起作用。



假如系统时钟使用晶体振荡器，则使用这种频率产生的方法可以产生非常精确的频率值。

预分频器(Prescaler)

TMRC 的位 0~2 可以用来定义定时/计数器中内部时钟源的预分频级数。定时/计数器溢出信号可用来驱动蜂鸣器或产生定时器的中断。

输入/输出接口

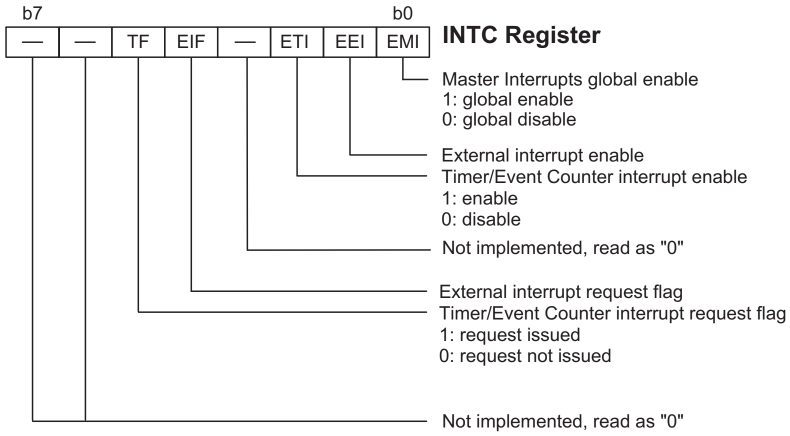
当运行在事件计数器或脉冲宽度测量模式时，定时/计数器需要使用外部引脚PC1/TMR 以确保正确的动作。由于 PC1/TMR 是共用引脚的，因此当这个引脚被配置为定时器输入引脚时，需选择正确的掩膜选项。这包括可以选择上拉电阻来连接定时器输入引脚。定时器也可被设定去驱动引脚共用的蜂鸣器。当通过掩膜选项选择蜂鸣器引脚时，定时器可以根据定时器预分频器(Prescaler)以及定时器 TMR 寄存器的内容，以不同的频率来驱动蜂鸣器。

编程注意事项

当定时/计数器运行在定时器模式时，定时器的时钟源是使用内部系统时钟，与单片机所有运算都能同步。在这个模式下，当定时器寄存器溢出时，单片机将产生一个内部中断信号，使程序进入相应的内部中断向量。对于脉冲宽度测量模式，定时器的时钟源也是使用内部系统时钟，但定时器只有在正确的逻辑条件出现在定时器输入引脚 PC1/TMR 时才执行动作。当这个外部事件没有和内部定时器时钟同步时，只有当下一个定时器时钟到达时，单片机才会看到这个外部事件，因此在测量值上可能有小的差异，需要程序设计者在程序应用时加以注意。同样的情况发生在定时器配置为外部事件计数模式时，它的时钟来源是外部事件，和内部系统时钟或者定时器时钟不同步。

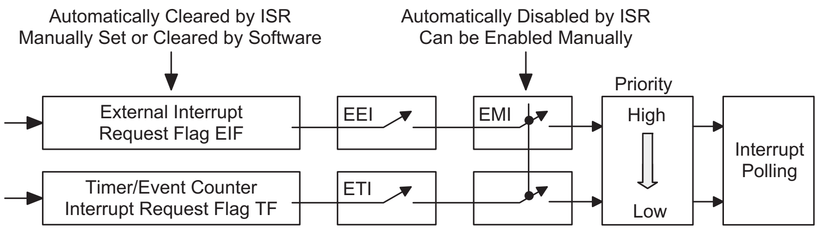
中断

单片机提供外部中断和内部定时/计数器中断两种功能。中断控制寄存器(INTC;0BH)包含设定中断打开/关闭的中断控制位和中断请求标志位。



一旦中断子程序被响应，所有其它的中断将被屏蔽(通过清零 EMI 位)。这个方式可以预防任何进一步的中断相互嵌套。其它的中断请求可能发生在这个期间，但只有中断请求标志位会被记录。如果某个中断服务子程序正在执行，此时有另一个中断要求响应，EMI 位和 INTC 相关的位可以被置位，以允许此中断响应。如果堆栈已满，即使此中断已经被允许，中断请求也不会响应，直到 SP 减少为止。如果要求立刻动作，堆栈必须避免成为储满状态。

单片机中所有中断都有唤醒暂停模式的能力。当一个中断被响应时，首先将程序计数器的值压入堆栈，并跳转到程序存储器中指定的子程序地址。要注意的是只有程序计数器的值会被压入堆栈。假如寄存器或状态寄存器的内容被中断服务程序改变，可能会破坏原先想要的控制顺序，因此这内容应该预先加已储存。



外部中断

外部中断是通过$\overline{\text{INT}}$端口上由高到低的电平转换来触发，之后相应的中断请求标志位(EIF；INTC 的位 4)将被置位。当外部中断被允许，堆栈没有满且外部中断产生时，将调用位于地址 04H 的子程序，中断请求标志位 EIF 被清零，且 EMI 位被清零来屏蔽其它中断。

定时/计数器中断

当定时/计数器中断请求标志位(TF；INTC 的位 5)因为定时器溢出而设定时，定时/计数器中断发生。当中断被允许、堆栈没有满且 TF 位被设定时，将调用位于地址 08H 的子程序，相关的中断请求标志位 TF 被清零，且 EMI 位被清零来屏蔽其它中断。

中断优先权

当中断是发生在两个连续的 T2 脉冲上升沿之间时，如果相应的中断请求被允许，中断将在后一个 T2 脉冲响应。下表指出在同时提出请求的情况下所提供的优先权，这个可以通过重新设定 EMI 位来加以屏蔽。

中断源	优先级	中断向量
外部中断	1	04H
定时/计数器中断	2	08H

中断优先权

当外部和内部两个中断同时被允许，且外部和内部中断同时发生的情况下，外部中断将拥有优先权首先动作。使用 INTC 寄存器来做个别中断的屏蔽可以预防同时发生的情形。

编程注意事项

定时/计数器中断请求标志位 TF、外部中断请求标志位 EIF、定时/计数器中断允许位 ETI、外部中断允许位 EEI 和主中断允许位 EMI 构成一个位于数据存储器 0BH 的中断控制寄存器 INTC。EMI、EEI 和 ETI 被用来控制中断的允许与否。当不允许中断时，这些位可以禁止中断响应动作。当中断请求标志位 TF 或 EIF 被设定，它们被保留在 INTC 寄存器直到中断服务子程序执行或被软件指令清零。

在中断子程序中建议不要使用“调用子程序”指令。中断通常发生在不可预期的情况或需要立刻执行的某些应用。假如只剩下一层堆栈且没有控制好中断使能，当“CALL”指令在中断子程序执行时，将破坏原来的控制序列。

复位和初始化

复位功能是任何单片机中基本的部分，使得单片机可以设定一些与外部参数无关的先置条件。最重要的是，单片机上电后，经短暂延迟，将处于预期的稳定状态并且准备执行第一条程序语句。上电复位之后，在程序未开始执行前，部分重要的内部寄存器将会被预先设定状态。程序计数器就是其中之一，它会被清除为零，使得单片机从最低的程序存储器地址开始执行程序。

除了上电复位外，即使单片机处于执行状态，有些情况的发生也迫使单片机必须复位。例如，当单片机上电执行程序后，$\overline{\text{RES}}$引脚被强制拉下至低电平。这个例子为正常操作复位，单片机中只有一些寄存器受影响，而大部份寄存器则是不受影响，以便复位引脚回复至高电平后，单片机仍可以正常操作。复位的另一种形式是看门狗定时器溢出而复位单片机，所有复位操作类型导致不同的寄存器条件被加以设定。

另外一种复位以低电压即 LVR 的形式存在，在电源提供电压低于某一临界值的情况下，一种和$\overline{\text{RES}}$引脚复位类似的完全复位将会被执行。

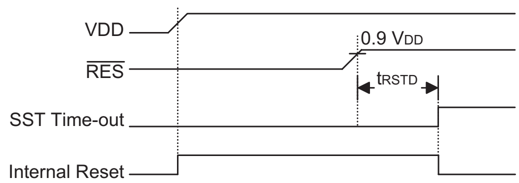
复位

包括内部与外部事件触发复位，单片机共有五种复位方式：

→ 上电复位

这是最基本而不可避免的复位，发生在单片机上电后。除了保证程序存储器会从起始地址开始执行，上电复位也使得其它寄存器被设定在预设条件，所有的输入/输出口寄存器和输入/输出口控制寄存器在上电复位时会保持高电平，以确保上电后所有引脚被设为输入状态。

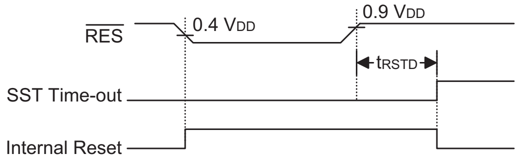
虽然单片机有一个内部 RC 复位功能，由于接通电源不稳定，还是推荐使用和$\overline{\text{RES}}$引脚连接的外部 RC 电路。由 RC 电路所造成的时间延迟使得$\overline{\text{RES}}$引脚在电源供应稳定前的一段延长周期内保持在低电平。在这段时间内，单片机是不能正常操作的。$\overline{\text{RES}}$引脚达到一定电压值后，再经过延迟时间 t_{RSTD}，单片机可开始进行正常操作。下图中 SST 是系统延迟周期（System Start-up Timer）的缩写。



上电复位时序图

→　**$\overline{RES}$引脚复位**

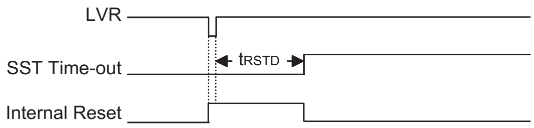
当单片机在正常工作，而$\overline{RES}$引脚通过外部硬件(如外部开关)被强迫拉至低电平时，此种复位形式即会发生。这种复位模式和其他复位的例子一样，程序计数器会被清除为零且程序从头开始执行。



$\overline{RES}$引脚复位时序图

→　**低电压复位－LVR**

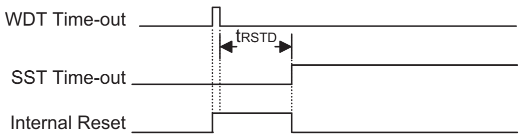
单片机有低电压复位电路的目的是为了监看单片机的电压供应源。例如更换电池的情况下，单片机供应的电压可能会落在 $0.9V\sim V_{LVR}$ 的范围内，这时 LVR 将会自动从内部复位单片机。有效的 LVR 信号，即 $0.9V\sim V_{LVR}$ 的低电压，必须存在超过 1ms。如果低电压存在不超过 1ms，则 LVR 将会忽略它且不会执行复位功能。



低电压复位时序图

→　**正常操作时看门狗溢出复位**

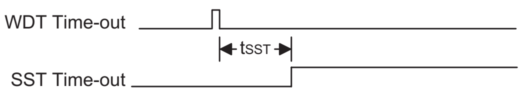
除了看门狗溢出标志位 TO 将被设为 1 之外，正常操作时看门狗溢出复位和$\overline{RES}$复位相同。



正常操作时看门狗溢出复位时序图

→　**暂停时看门狗溢出复位**

暂停时看门狗溢出复位有些不同于其他种类的复位，除了程序计数器与堆栈指针将被清除为 0 及 TO 标志位被设为 1 外，绝大部份的条件保持不变。有关 t_{SST} 细节请见 A.C 特性。



暂停时看门狗溢出复位时序图

不同的复位方法以不同的方式影响复位标志位。这些标志位即 PDF 和 TO，被放在状态寄存器中，由如暂停功能或看门狗计数器等几种控制器操作加以控制。复位标志位如下所示：

TO	PDF	复位条件
0	0	上电时的 $\overline{RES}$ 复位
u	u	一般运行时的 $\overline{RES}$ 复位或 LVR 低压复位
1	u	一般运行时的 WDT 溢出复位
1	1	HALT 暂停时的 WDT 溢出复位

“u” 表示不变化

在单片机复位之后，各功能单元初始化的情形，列于下表。

项 目	复位后情况
程序计数器	清除为零
中断	所有中断被关闭
看门狗定时器	WDT 清零并重新计时
定时/计数器	所有定时/计数器停止
预分频器	定时/计数器之预分频器内容清零
输入/输出口	所有 I/O 设为输入模式
堆栈指针	堆栈指针指向堆栈顶端

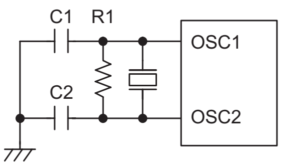
不同的复位以不同的方式影响单片机中的内部寄存器。为保证复位发生后程序的正常执行，在特定的复位发生后，了解单片机内的情况是非常重要的。下表是描述不同复位是如何影响单片机的内部寄存器。

寄存器	$\overline{RES}$ 复位 (上电时)	$\overline{RES}$ 或 LVR 复位 (一般运行时)	WDT 溢出复位 (一般运行时)	WDT 溢出复位 (HALT 暂停时)
MP	– x x x x x x x	– u u u u u u u	– u u u u u u u	– u u u u u u u
ACC	x x x x x x x x	u u u u u u u u	u u u u u u u u	u u u u u u u u
PCL	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
TBLP	x x x x x x x x	u u u u u u u u	u u u u u u u u	u u u u u u u u
TBLH	– – x x x x x x	– – u u u u u u	– – u u u u u u	– – u u u u u u
STATUS	– – 0 0 x x x x	– – u u u u u u	– – 1 u u u u u	– – 1 1 u u u u
INTC	– – 0 0 – 0 0 0	– – 0 0 – 0 0 0	– – 0 0 – 0 0 0	– – u u – u u u
WDTS	0 0 0 0 0 1 1 1	0 0 0 0 0 1 1 1	0 0 0 0 0 1 1 1	u u u u u u u u
TMR	x x x x x x x x	x x x x x x x x	x x x x x x x x	u u u u u u u u
TMRC	0 0 – 0 1 0 0 0	0 0 – 0 1 0 0 0	0 0 - 0 1 0 0 0	u u – u u u u u
PA	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	u u u u u u u u
PAC	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	u u u u u u u u
PB	– – – – – 1 1 1	– – – – – 1 1 1	– – – – – 1 1 1	– – – – – u u u
PBC	– – – – – 1 1 1	– – – – – 1 1 1	– – – – – 1 1 1	– – – – – u u u
PC	– – – – – – 1 1	– – – – – – 1 1	– – – – – – 1 1	– – – – – – u u
PCC	– – – – – – 1 1	– – – – – – 1 1	– – – – – – 1 1	– – – – – – u u

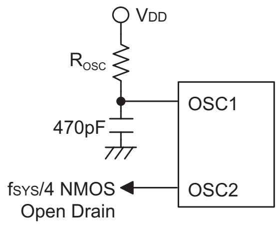
“u”表示不变化
“x”表示不确定
“–”表示不存在

振荡器

有两种方法产生系统时钟：使用外部晶体/陶瓷振荡器或是外部 RC 电路。选择方法是通过掩膜选项来加以选择。参考下图。



晶体/陶瓷振荡器



电阻电容振荡器

下表表示对于几种晶体/陶瓷的振荡频率下的典型的 C1、C2 和 R1 的值。对外部 RC 振荡器 R_{OSC} 请参考附录章节中 RC 振荡器与温度及 V_{DD} 特性曲线。

晶体或共振器	C1, C2	R1
4MHz 晶体	0pF	10kΩ
4MHz 共振器（3 脚）	0pF	12kΩ
4MHz 共振器（2 脚）	10pF	12kΩ
3.58MHz 晶体	0pF	10kΩ
3.58MHz 共振器（2 脚）	25pF	10kΩ
2MHz 晶体与共振器（2 脚）	25pF	10kΩ
1MHz 晶体	35pF	27kΩ
480kHz 共振器	300pF	9.1kΩ
455kHz 共振器	300pF	10kΩ
429kHz 共振器	300pF	10kΩ

系统晶体/陶瓷振荡器

对于晶体振荡器的结构配置，晶体只要简单的连接至 OSC1 和 OSC2，则会产生所需的相移及反馈，而不需其他外部的器件。陶瓷共振器可以用来代替晶体，但是要连接两个小电容在 OSC1、OSC2 和地之间。

系统电阻电容振荡器

使用外部 RC 电路作为系统振荡器，需要在 OSC1 和 VDD 之间连接一个阻值在 24kΩ到 1MΩ之间的电阻，且要连接一个 470pF 的电容到地。系统时钟 4 分频后提供给 OSC2 作输出，以达到外部同步化的目的。虽然此振荡器配置成本较低，但振荡频率会因 VDD、温度和芯片本身的制成而改变，因此不适合用来做计时严格或需要精确振荡器频率的场合。

看门狗定时振荡器

WDT 振荡器是一种完全独立在芯片上自由动作的振荡器，它在 5V 时的周期时间典型值 65μs，不需外部的器件搭配。当芯片进入暂停模式时，系统时钟将停止动作，但 WDT 振荡器继续自由动作且保持看门狗有效。然而在某些应用中，为了保持功率，WDT 振荡器可以通过掩膜选项来关闭。

暂停模式下的暂停和唤醒

暂停模式是通过“HALT”指令实现且造成如下结果：

- 系统振荡器将被关闭
- 在 RAM 芯片和寄存器上的内容保持不变
- WDT 和 WDT 预分频器(Prescaler)被清零，假如 WDT 时钟源是来自 WDT 振荡器，那么会重新计数
- 所有输入/输出端口状态保持不变
- PDF 标志位被置位而 TO 标志位被清零

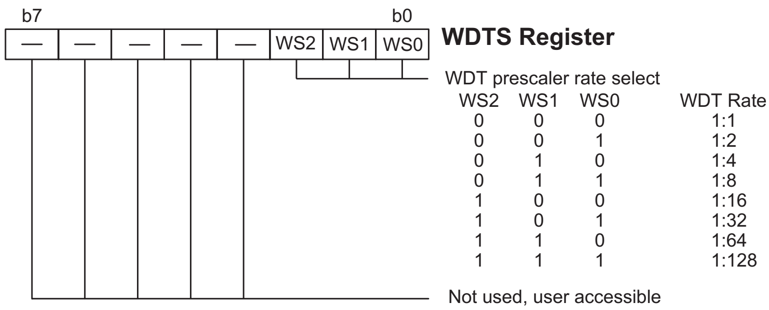
系统可以通过外部复位、内部中断、端口 PA 上外部下降沿信号或 WDT 溢出来离开暂停模式。外部复位将初始化芯片，而 WDT 溢出也可以从暂停模式启动 WDT 溢出复位，通过检查 TO 和 PDF 标志位，可以判定复位来源。PDF 标志位可以通过系统上电或执行“CLR WDT”指令来清零，而执行“HALT”指令则可置位 PDF 标志位。如果 WDT 溢出发生，TO 标志位将被置位，同时造成一个只复位 PC 和 SP 的唤醒，而其它的标志位则保持原来状态。

端口 PA 唤醒和中断唤醒方法可以视为一般执行的延续。端口 PA 中的每个位可以通过掩膜选项独立选择唤醒功能。输入/输出口唤醒后，程序将在下一条指令处继续执行。如果系统是通过中断唤醒，则有两种可能，假如相关的中断被禁止或中断允许但堆栈已满，程序将在下一条指令处继续执行；假如相关的中断允许且堆栈未满，则正常的中断响应将会发生。假设在进入暂停模式之前中断请求标志位被设为 1，相关中断的唤醒功能将无效。一旦唤醒事件发生，回到正常运算将需要 1024 个系统时钟周期。换句话说，一个空周期将插在唤醒之后。如果在唤醒后是接着去响应中断，则真实的中断子程序执行将延迟一个或数个周期，如果唤醒后接着去执行下一条指令，则它将在若干空周期结束后立刻执行。

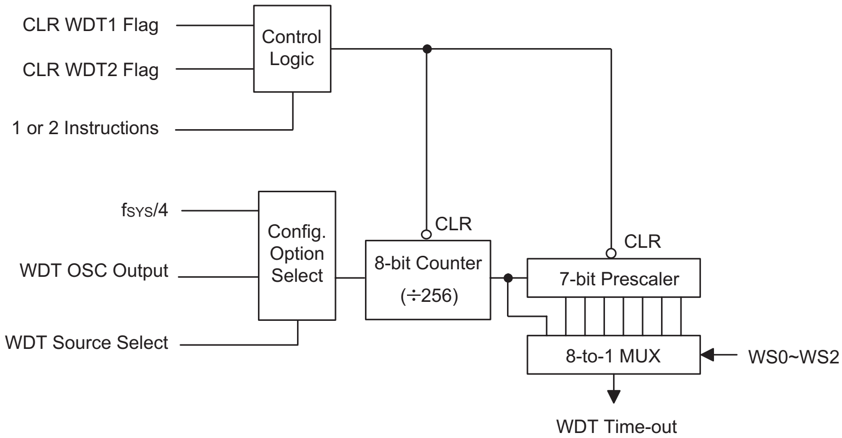
看门狗定时器

看门狗定时器的功能在防止如电的干扰等外部不可控制事件，所造成的程序不正常动作或跳转到未知的地址。当 WDT 溢出时，它产生一个“芯片复位”的动作。WDT 时钟通过选择掩膜选项中两个时钟源之一提供：它本身内部的 WDT 振荡器或指令时钟(系统时钟除以 4)。要注意的是假如 WDT 掩膜选项设为除能，则任何相关的指令将无效。

内部 WDT 振荡器在 5V 电压下周期为 65μs 左右。如果选择这种操作条件，它首先通过 8 阶计数器除以 256 来产生一个 17ms 的周期，要注意的是这个周期会因 VDD、温度和制成而改变。WDT 预分频器(Prescaler)可以用于产生更长的溢出周期，通过写入需要的值到 WDT 寄存器位 0、1 和 2，即 WS0、WS1 和 WS2，更长的溢出周期即能被完成。如果 WS0、WS1 和 WS2 都等于 1，分频比例为 1∶128 且产生大约 2.1s 的最大溢出周期。WDT 中高半字节和位 3 保留给使用者定义标志位，可以用来指示一些特殊状况。



WDT 振荡器可以被关闭，且 WDT 时钟可以通过指令时钟(系统时钟除以 4)所提供，假如指令时钟作为时钟源，要注意到当系统进入到暂停模式时，指令时钟会停止且 WDT 将失去其保护作用，在这种情况下系统只能通过外部逻辑重新打开，当系统操作在干扰严重的环境时，建议使用内部 WDT 振荡器。



看门狗定时器

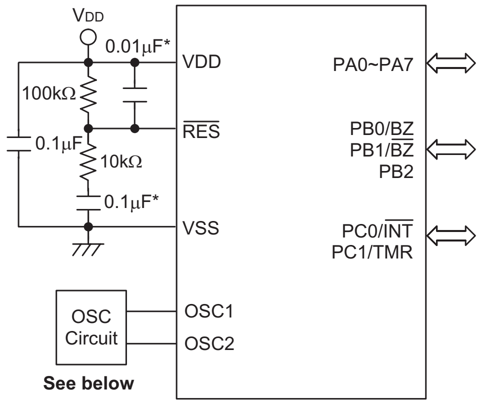
系统在正常运行状态下，WDT 溢出将导致“芯片复位”，且置位状态位“TO”。然而假如系统在暂停模式，WDT 溢出复位发生，它只影响程序计数器和 SP。有三种方法可以用来清除 WDT 的内容，包括 WDT 预分频器(Prescaler)。第一种是外部硬件复位($\overline{RES}$ 引脚低电平)，第二种是通过软件指令，而第三种是通过 HALT 指令。使用软件指令有两种方法去清除看门狗寄存器，必须由掩膜选项选择。第一种选择是使用单一“CLR WDT”指令而第二种是使用“CLR WDT1”和“CLR WDT2”两条指令。对于第一种选择，只要执行“CLR WDT”便清除 WDT。而第二种选择必须交替执行“CLR WDT1”和“CLR WDT2”两者才能成功的清除 WDT。关于第二种选择要注意到如果“CLR WDT1”正被使用来清除 WDT，接着执行这条指令是无效的，只有执行“CLR WDT2”指令能清除 WDT。同样地“CLR WDT2”指令已经执行后，只有接着执行“CLR WDT1”指令才可以清除看门狗定时器。

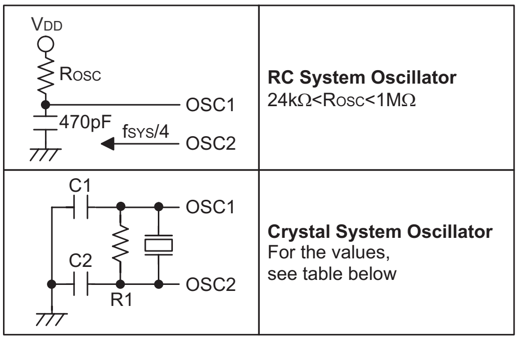
掩膜选项

通过 HT-IDE 的软件介面，使用者可以选择该系列所有的单片机掩膜选项。对于 OTP 版单片机，单片机掩膜选项会被储存在选项存储器。对于掩膜版单片机，单片机掩膜选项则会于工厂生产时制作完成。所有的位必须按照适合的系统功能去设定，具体内容可由下表得到。

No.	选项
1	WDT 时钟来源：WDT OSC 或 $f_{SYS}/4$
2	WDT 功能：打开/关闭
3	LVR 功能：打开/关闭
4	CLRWDT 指令条数：1 / 2 条
5	系统振荡器: RC 或晶体振荡
6	上拉电阻(PA~PC)：无/有
7	BZ 功能：打开/关闭
8	PA0~PA7 唤醒功能：打开/关闭

应用电路





OSC Circuit

第二部份

程序语言

第二章

指令集介绍

2

指令集

任何单片机成功运作的核心在于它的指令集，此指令集为一组程序指令码，用来指导单片机如何去执行特定的工作。在盛群单片机中，提供了丰富且易变通的指令，共超过六十条，程序设计师可以事半功倍地实现他们的应用。

为了更加容易了解各式各样的指令码，接下来按功能分组介绍它们。

指令周期

大部分指令只需要一个指令周期来执行。分支、调用或查表则需要两个指令周期。一个指令周期相当于四个系统时钟周期，因此如果在 8MHz 的系统时钟振荡器下，大部分的操作将在 0.5μs 中执行完成，而分支或调用操作则将在 1μs 中执行完成。虽然需要两个指令周期的指令通常是指的是 JMP、CALL、RET、RETI 和查表指令，但如果指令牵涉到程序计数器低字节寄存器 PCL 也将多花费一个周期去加以执行。一旦指令改变 PCL 的内容，进而导致直接跳转至新地址时，就需要多一个周期去执行。例如“CLR PCL”或“MOV PCL, A”。对于跳转命令必须注意的是，如果比较的结果牵涉到跳转动作将多花费一个周期，如果没有则需一个周期即可。

数据的传送

单片机程序中数据传送是使用最为频繁的操作之一。使用三种 MOV 的指令，数据不但可以从寄存器转移至累加器(反之亦然)，而且能够直接传送立即数到累加器。数据传送最重要的应用之一是从接收端口接收数据或者传送数据到输出端口。

算术运算

算术运算和数据处理是大部分单片机应用所需具备的能力，在盛群单片机内部的指令集中，可直接实现加与减的运算。当加法的结果超出 255 或减法的结果少于 0 时，要注意正确的处理进位和借位的问题。INC、INCA、DEC 和 DECA 指令提供了对一个指定地址的值加一或减一的功能。

逻辑和移位运算

标准逻辑运算例如 AND、OR、XOR 和 CPL 全都包含在盛群单片机内部的指令集中。大多数牵涉到数据运算的指令，数据的传送必须通过累加器。在所有逻辑数据运算中，如果运算结果为零，则零标志位将被置位。另外逻辑数据运用形式还有移位指令，例如 RR、RL、RRC 和 RLC 提供了向左或向右移动一位的方法。移位指令常用于串行端口的程序应用，数据可从内部寄存器转移至进位标志位，而此位则可被检验。移位运算还可应用在乘法与除法的运算组成中。

分支和控制的转换

程序分支是采取使用 JMP 指令跳转至指定地址或使用 CALL 指令调用子程序的形式。两者之不同在于当子程序被执行完毕后，程序必须马上返回原来的地址。这个动作是由放置在子程序里的返回指令 RET 来实现，它可使程序跳回 CALL 指令之后的地址。在 JMP 指令中，程序则只是跳到一个指定的地址而已，并不需如 CALL 指令般跳回。一个非常有用的分支指令是条件跳转，跳转条件是由数据存储器或指定位来加以决定。遵循跳转条件，程序将 继续执行下一条指令或略过且跳转至接下来的指令。这些分支指令是程序走向的关键，跳转条件可能是外部开关输入，或者内部数据位的值。

位运算

提供数据存储器中单个位的运算指令是盛群单片机的特性之一。这特性对于输出端口位的规划尤其有用，其中个别的位或端口的引脚可以使用 “SET [m].i” 或“CLR [m].i”指令来设定其为高位或低位。如果没有这特性，程序设计师必须先读入输出口的 8 位数据，处理这些数据，然后再输出正确的新数据。此种读入-修改-写出的过程现在则被位运算指令所取代。

查表运算

数据的储存通常由寄存器完成。然而当处理大量固定的数据时，它的存储量常常造成对个别存储器的不便。为了改善此问题，盛群单片机允许在程序存储器中设定一块可直接存储数据的区域，只需要一组简易的指令即可对数据进行查表。

其它运算

除了上述功能指令外，其他指令还包括用于暂停的“HALT”指令和使程序在极端电压或电磁环境下仍能正常工作的看门狗定时器控制指令。这些指令的使用请查阅相关的章节。

指令设定一览表

惯例

x：立即数
m：数据存储器地址
A：累加器
i：0~7 号位
addr：程序存储器地址

助记符	指令简易描述	周期	影响标志位
算术运算			
ADD A, [m]	ACC 与数据存储器相加，结果放入 ACC	1	Z,C,AC,OV
ADDM A, [m]	ACC 与数据存储器相加，结果放入数据存储器	1[Note]	Z,C,AC,OV
ADD A, x	ACC 与立即数相加，结果放入 ACC	1	Z,C,AC,OV
ADC A, [m]	ACC 与数据存储器、进位标志相加，结果放入 ACC	1	Z,C,AC,OV
ADCM A, [m]	ACC 与数据存储器、进位标志相加，结果放入数据存储器	1[Note]	Z,C,AC,OV
SUB A, x	ACC 与立即数相减，结果放入 ACC	1	Z,C,AC,OV
SUB A, [m]	ACC 与数据存储器相减，结果放入 ACC	1	Z,C,AC,OV
SUBM A, [m]	ACC 与数据存储器相减，结果放入数据存储器	1[Note]	Z,C,AC,OV
SBC A, [m]	ACC 与数据存储器、进位标志相减，结果放入 ACC	1	Z,C,AC,OV
SBCM A, [m]	ACC 与数据存储器、进位标志相减，结果放入数据存储器	1[Note]	Z,C,AC,OV
DAA [m]	将加法运算中放入 ACC 的值调整为十进制数，并将结果放入数据存储器	1[Note]	C

助记符	指令简易描述	周期	影响标志位
逻辑运算			
AND A, [m]	ACC 与数据存储器做“与”运算，结果放入 ACC	1	Z
OR A, [m]	ACC 与数据存储器做“或”运算，结果放入 ACC	1	Z
XOR A, [m]	ACC 与数据存储器做“异或”运算，结果放入 ACC	1	Z
ANDM A, [m]	ACC 与数据存储器做“与”运算，结果放入数据存储器	1[Note]	Z
ORM A, [m]	ACC 与数据存储器做“或”运算，结果放入数据存储器	1[Note]	Z
XORM A, [m]	ACC 与数据存储器做“异或”运算，结果放入数据存储器	1[Note]	Z
AND A, x	ACC 与立即数做“与”运算，结果放入 ACC	1	Z
OR A, x	ACC 与立即数做“或”运算，结果放入 ACC	1	Z
XOR A, x	ACC 与立即数做“异或”运算，结果放入 ACC	1	Z
CPL [m]	对数据存储器取反，结果放入数据存储器	1[Note]	Z
CPLA [m]	对数据存储器取反，结果放入 ACC	1	Z
递增和递减			
INCA [m]	递增数据存储器，结果放入 ACC	1	Z
INC [m]	递增数据存储器，结果放入数据存储器	1[Note]	Z
DECA [m]	递减数据存储器，结果放入 ACC	1	Z
DEC [m]	递减数据存储器，结果放入数据存储器	1[Note]	Z
移位			
RRA [m]	数据存储器右移一位，结果放入 ACC	1	无
RR [m]	数据存储器右移一位，结果放入数据存储器	1[Note]	无
RRCA [m]	带进位将数据存储器右移一位，结果放入 ACC	1	C
RRC [m]	带进位将数据存储器右移一位，结果放入数据存储器	1[Note]	C
RLA [m]	数据存储器左移一位，结果放入 ACC	1	无
RL [m]	数据存储器左移一位，结果放入数据存储器	1[Note]	无
RLCA [m]	带进位将数据存储器左移一位，结果放入 ACC	1	C
RLC [m]	带进位将数据存储器左移一位，结果放入数据存储器	1[Note]	C
数据传送			
MOV A, [m]	将数据存储器送至 ACC	1	无
MOV [m], A	将 ACC 送至数据存储器	1[Note]	无
MOV A, x	将立即数送至 ACC	1	无
位运算			
CLR [m].i	清零数据存储器的位	1[Note]	无
SET [m].i	置位数据存储器的位	1[Note]	无

助记符	指令简易描述	周期	影响标志位
转移			
JMP addr	无条件跳转	2	无
SZ [m]	如果数据存储器为零，则跳过下一条指令	1 Note	无
SZA [m]	数据存储器送至 ACC，如果内容为零，则跳过下一条指令	1 Note	无
SZ [m].i	如果数据存储器的第 i 位为零，则跳过下一条指令	1 Note	无
SNZ [m].i	如果数据存储器的第 i 位不为零，则跳过下一条指令	1 Note	无
SIZ [m]	递增数据存储器，如果结果为零，则跳过下一条指令	1 Note	无
SDZ [m]	递减数据存储器，如果结果为零，则跳过下一条指令	1 Note	无
SIZA [m]	递增数据存储器，将结果放入 ACC，如果结果为零，则跳过下一条指令	1 Note	无
SDZA [m]	递减数据存储器，将结果放入 ACC，如果结果为零，则跳过下一条指令	1 Note	无
CALL addr	子程序调用	2	无
RET	从子程序返回	2	无
RET A, x	从子程序返回，并将立即数放入 ACC	2	无
RETI	从中断返回	2	无
查表			
TABRDC [m]	读取当前页的 ROM 内容，并送至数据存储器和 TBLH	2 Note	无
TABRDL [m]	读取最后页的 ROM 内容，并送至数据存储器和 TBLH（该指令对 HT48R05A-1/HT48C05 无效）	2 Note	无
其它指令			
NOP	空指令	1	无
CLR [m]	清零数据存储器	1 Note	无
SET [m]	置位数据存储器	1 Note	无
CLR WDT	清零看门狗定时器	1	TO,PDF
CLR WDT1	预清除看门狗定时器	1	TO,PDF
CLR WDT2	预清除看门狗定时器	1	TO,PDF
SWAP [m]	交换数据存储器的高低字节，结果放入数据存储器	1 Note	无
SWAPA [m]	交换数据存储器的高低字节，结果放入 ACC	1	无
HALT	进入暂停模式	1	TO,PDF

注意：
1. 对跳转指令而言，如果比较的结果牵涉到跳转即需 2 个周期，如果没有跳转发生，则只需一个周期即可。
2. 任何指令若要改变 PCL 的内容将需要 2 个周期来执行。
3. 对于“CLR WDT1”和“CLR WDT2”指令而言， TO 和 PDF 标志位也许会受执行结果影响。“CLR WDT1”和“CLR WDT2”被执行后，TO 和 PDF 标志位会被清零，除此之外 TO 和 PDF 标志位保持不变。

第三章

指令定义

3

ADC A, [m]　Add Data Memory to ACC with Carry

指令说明　将指定数据存储器、累加器和进位标志位的内容相加后，把结果储存回累加器。

功能表示　ACC ← ACC + [m] + C

影响标志位　OV , Z , AC , C

ADCM A, [m]　Add ACC to Data Memory with Carry

指令说明　将指定数据存储器、累加器和进位标志位的内容相加后，把结果储存回指定数据存储器。

功能表示　[m] ← ACC + [m] + C

影响标志位　OV , Z , AC , C

ADD A, [m]　Add Data Memory to ACC

指令说明　将指定数据存储器和累加器的内容相加后，把结果储存回累加器。

功能表示　ACC ← ACC + [m]

影响标志位　OV , Z , AC , C

ADD A, x　Add immediate data to ACC

指令说明　将累加器和立即数的内容相加后，把结果储存回累加器。

功能表示　ACC ← ACC + x

影响标志位　OV , Z , AC , C

ADDM A, [m] Add ACC to Data Memory
指令说明 将指定数据存储器和累加器的内容相加后，把结果储存回指定数据存储器。
功能表示 [m] ← ACC + [m]
影响标志位 OV , Z , AC , C

AND A, [m] Logical AND Data Memory to ACC
指令说明 将存在累加器和指定数据存储器中的数据作 AND 的运算，然后把结果储存回累加器。
功能表示 ACC ← ACC“AND”[m]
影响标志位 Z

AND A, x Logical AND immediate data to ACC
指令说明 将存在累加器中的数据和立即数作 AND 的运算，然后把结果储存回累加器。
功能表示 ACC ← ACC“AND”x
影响标志位 Z

ANDM A, [m] Logical AND ACC to Data Memory
指令说明 将存在指定数据存储器和累加器中的数据作 AND 的运算，然后把结果储存回数据存储器。
功能表示 [m] ← ACC“AND”[m]
影响标志位 Z

CALL addr Subroutine call
指令说明 无条件地调用指定地址的子程序，此时程序计数器先加 1 获得下一个要执行的指令地址并压入堆栈，接着载入指定地址并从新地址继续执行程序，由于此指令需要额外的运算，所以为一个 2 周期的指令。
功能表示 Stack ← Program Counter + 1
Program Counter ← addr
影响标志位 None

CLR [m] Clear Data Memory
指令说明 指定数据存储器中的每一位均清除为 0。
功能表示 [m] ← 00H
影响标志位 None

CLR [m].i　Clear bit of Data Memory
指令说明　指定数据存储器中的 i 位清除为 0。
功能表示　[m].i ← 0
影响标志位　None

CLR WDT　Clear Watchdog Timer
指令说明　将 TO、PDF 标志位和 WDT 全都清零。
功能表示　WDT cleared
TO ← 0
PDF ← 0
影响标志位　TO , PDF

CLR WDT1　Pre-clear Watchdog Timer
指令说明　将 TO、PDF 标志位和 WDT 全都清零，请注意此指令要结合 CLR WDT2 一起动作且必须交替执行才有作用，重复执行此项指令而没有与 CLR WDT2 交替执行将无任何作用。
功能表示　WDT cleared
TO ← 0
PDF ← 0
影响标志位　TO , PDF

CLR WDT2　Pre-clear Watchdog Timer
指令说明　将 TO、PDF 标志位和 WDT 全都清零，请注意此指令要结合 CLR WDT1 一起动作且必须交替执行才有作用，重复执行此项指令而没有与 CLR WDT1 交替执行将无任何作用。
功能表示　WDT cleared
TO ← 0
PDF ← 0
影响标志位　TO , PDF

CPL [m]　Complement Data Memory
指令说明　将指定数据存储器中的每一位取逻辑反，相当于从 1 变 0 或 0 变 1。
功能表示　[m] ← $\overline{[m]}$
影响标志位　Z

CPLA [m] Complement Data Memory with result in ACC

指令说明 将指定数据存储器中的每一位取逻辑反，相当于从 1 变 0 或 0 变 1，而结果被储存回累加器且数据存储器中的内容不变。

功能表示 ACC ← $\overline{[m]}$

影响标志位 Z

DAA [m] Decimal-Adjust ACC for addition with result in Data Memory

指令说明 将存在累加器中的内容数值转换为 BCD（二进制转成十进制）数值，如果低 4 位大于 9 或 AC 标志位被置位，则在低 4 位加上一个 6，不然低 4 位的内容不变，如果高 4 位大于 9 或 C 标志位被置位，则在高 4 位加上一个 6，十进制的转换主要是依照累加器和标志位状况，分别加上 00H、06H、60H 或 66H，只有 C 标志位也许会被此指令影响，它会指出原始 BCD 数是否大于 100，并可以进行双精度十进制数据相加。

功能表示 [m] ← ACC + 00H 或
[m] ← ACC + 06H 或
[m] ← ACC + 60H 或
[m] ← ACC + 66H

影响标志位 C

DEC [m] Decrement Data Memory

指令说明 将在指定数据存储器内的数据减 1。

功能表示 [m] ← [m] – 1

影响标志位 Z

DECA [m] Decrement Data Memory with result in ACC

指令说明 将在指定数据存储器内的数据减 1，把结果储存回累加器且数据存储器中的内容不变。

功能表示 ACC ← [m] – 1

影响标志位 Z

HALT Enter power down mode
指令说明 此指令停止程序的执行并且关闭系统时钟，但数据存储器和寄存器的内容仍被保留，WDT 和预分频器(Prescaler)被清零，暂停标志位 PDF 被置位且 WDT 溢出标志位 TO 被清零。
功能表示 TO ← 0
PDF ← 1
影响标志位 TO , PDF

INC [m] Increment Data Memory
指令说明 将指定数据存储器内的数据加 1。
功能表示 [m] ← [m] + 1
影响标志位 Z

INCA [m] Increment Data Memory with result in ACC
指令说明 将指定数据存储器内的数据加 1，把结果储存回累加器且数据存储器中的内容不变。
功能表示 ACC ← [m] + 1
影响标志位 Z

JMP addr Jump unconditionally
指令说明 程序计数器的内容被指定地址所取代，程序由新地址继续执行，当新地址被载入时，必须插入一虚指令，所以此指令为 2 个周期的指令
功能表示 Program Counter ← addr
影响标志位 None

MOV A, [m] Move Data Memory to ACC
指令说明 将指定数据存储器的内容复制到累加器中。
功能表示 ACC ← [m]
影响标志位 None

MOV A, x Move immediate data to ACC
指令说明 将立即数载入至累加器中。
功能表示 ACC ← x
影响标志位 None

MOV [m], A Move ACC to Data Memory
指令说明 将累加器的内容复制到指定数据存储器。
功能表示 [m] ← ACC
影响标志位 None

NOP No operation
指令说明 空操作，接下来顺序执行下一条指令。
功能表示 No operation
影响标志位 None

OR A, [m] Logical OR Data Memory to ACC
指令说明 将存在累加器和指定数据存储器中的数据作 OR 的运算，然后把结果储存回累加器。
功能表示 ACC ← ACC“OR”[m]
影响标志位 Z

OR A, x Logical OR immediate data to ACC
指令说明 将存在累加器中的数据和立即数作 OR 的运算，然后把结果储存回累加器。
功能表示 ACC ← ACC“OR”x
影响标志位 Z

ORM A, [m] Logical OR ACC to Data Memory
指令说明 将存在指定数据存储器和累加器中的数据作 OR 的运算，然后把结果储存回数据存储器。
功能表示 [m] ← ACC“OR”[m]
影响标志位 Z

RET Return from subroutine
指令说明 将堆栈区的数据取回至程序计数器，程序由取回的地址继续执行。
功能表示 Program Counter ← Stack
影响标志位 None

RET A, x Return from subroutine and load immediate data to ACC

指令说明 将堆栈区的数据取回至程序计数器且累加器载入立即数，程序由取回的地址继续执行。

功能表示 Program Counter ← Stack

ACC ← x

影响标志位 None

RETI Return from interrupt

指令说明 将堆栈区的数据取回至程序计数器且中断功能通过 EMI 位重新被使能，EMI 是控制中断使能的主中断位(寄存器 INTC 的第 0 位)。如果在执行 RETI 指令之前还有中断未被响应，则这个中断将在返回主程序之前被响应。

功能表示 Program Counter ← Stack

EMI ← 1

影响标志位 None

RL [m] Rotate Data Memory left

指令说明 将指定数据存储器的内容向左移 1 个位，且第 7 位移回第 0 位。

功能表示 [m].(i+1) ← [m].i ; (i = 0~6)

[m].0 ← [m].7

影响标志位 None

RLA [m] Rotate Data Memory left with result in ACC

指令说明 将指定数据存储器的内容向左移 1 个位，且第 7 位移回第 0 位，而移位的结果储存回累加器且数据存储器中的内容不变。

功能表示 ACC.(i+1) ← [m].i ; (i = 0~6)

ACC.0 ← [m].7

影响标志位 None

RLC [m] Rotate Data Memory Left through Carry

指令说明 将指定数据存储器的内容连同进位标志位向左移 1 个位，第 7 位取代进位位且原本的进位标志位移至第 0 位。

功能表示 [m].(i+1) ← [m].i ; (i = 0~6)

[m].0 ← C

C ← [m].7

影响标志位 C

RLCA [m] Rotate Data Memory left through Carry with result in ACC

指令说明 将指定数据存储器的内容连同进位标志位向左移 1 个位，第 7 位取代进位位且原本的进位标志位移至第 0 位，而移位的结果储存回累加器且数据存储器中的内容不变。

功能表示 ACC.(i+1) ← [m].i ; (i = 0~6)

ACC.0 ← C

C ← [m].7

影响标志位 C

RR [m] Rotate Data Memory right

指令说明 将指定数据存储器的内容向右移 1 个位，且第 0 位移回第 7 位。

功能表示 [m].i ← [m].(i+1) ; (i = 0~6)

[m].7 ← [m].0

影响标志位 None

RRA [m] Rotate Data Memory right with result in ACC

指令说明 将指定数据存储器的内容向右移 1 个位，且第 0 位移回第 7 位，而移位的结果储存回累加器且数据存储器中的内容不变。

功能表示 ACC.i ← [m].(i+1) ; (i = 0~6)

ACC.7 ← [m].0

影响标志位 None

RRC [m] Rotate Data Memory right through Carry

指令说明 将指定数据存储器的内容连同进位标志位向右移 1 个位，第 0 位取代进位位且原本的进位标志位移至第 7 位。

功能表示 [m].i ← [m].(i+1) ; (i = 0~6)

[m].7 ← C

C ← [m].0

影响标志位 C

RRCA [m]　Rotate Data Memory right through Carry with result in ACC

指令说明　将指定数据存储器的内容连同进位标志位向右移 1 个位，第 0 位取代进位位且原本的进位标志位移至第 7 位，而移位的结果储存回累加器且数据存储器中的内容不变。

功能表示　ACC.i ← [m].(i+1) ; (i = 0~6)

ACC.7 ← C

C ← [m].0

影响标志位　C

SBC A, [m]　Subtract Data Memory from ACC with Carry

指令说明　将累加器中的数据与指定数据存储器内容和进位标志位的反相减，把结果储存回累加器。如果结果为负，C 标志位清为 0，反之结果为正或 0，C 标志位设置为 1。

功能表示　ACC ← ACC – [m] – $\overline{C}$

影响标志位　OV , Z , AC , C

SBCM A, [m]　Subtract Data Memory from ACC with Carry and result in Data Memory

指令说明　将累加器中的数据与指定数据存储器内容和进位标志位的反相减，把结果储存回数据存储器。如果结果为负，C 标志位清为 0，反之结果为正或 0，C 标志位设置为 1。

功能表示　[m] ← ACC – [m] – $\overline{C}$

影响标志位　OV , Z , AC , C

SDZ [m]　Skip if Decrement Data Memory is 0

指令说明　将指定数据存储器的内容先减去 1 后，如果结果为 0，则程序计数器再加 1 跳过下一条指令，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，则程序继续执行下面的指令。

功能表示　[m] ← [m] – 1

Skip if [m] = 0

影响标志位　None

SDZA [m] Skip if decrement Data Memory is zero with result in ACC

指令说明 将指定数据存储器的内容先减去 1 后，如果结果为 0，则程序计数器再加 1 跳过下一条指令，此结果会被储存回累加器且指定数据存储器中的内容不变，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，则程序继续执行下面的指令。

功能表示 ACC ← [m] – 1

Skip if ACC = 0

影响标志位 None

SET [m] Set Data Memory

指令说明 将指定数据存储器的每一个位置位为 1。

功能表示 [m] ← FFH

影响标志位 None

SET [m].i Set bit of Data Memory

指令说明 将指定数据存储器的第 i 位置位为 1。

功能表示 [m].i ← 1

影响标志位 None

SIZ [m] Skip if increment Data Memory is 0

指令说明 将指定数据存储器的内容先加上 1 后，如果结果为 0，则程序计数器再加 1 跳过下一条指令，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，则程序继续执行下面的指令。

功能表示 [m] ← [m] + 1

Skip if [m] = 0

影响标志位 None

SIZA [m] Skip if increment Data Memory is zero with result in ACC

指令说明 将指定数据存储器的内容先加上 1 后，如果结果为 0，则程序计数器再加 1 跳过下一条指令，此结果会被储存回累加器且指定数据存储器中的内容不变，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，则程序继续执行下面的指令。

功能表示 ACC ← [m] + 1

Skip if ACC = 0

影响标志位 None

SNZ [m].i Skip if bit i of Data Memory is not 0

指令说明 如果指定数据存储器的第 i 位不为 0，则程序计数器再加 1 跳过下一条指令，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，程序继续执行下面的指令。

功能表示 Skip if [m].i ≠ 0

影响标志位 None

SUB A, [m] Subtract Data Memory from ACC

指令说明 将累加器中内容减去指定数据存储器的数据，把结果储存回累加器。如果结果为负，C 标志位清为 0，反之结果为正或 0，C 标志位设置为 1。

功能表示 ACC ← ACC – [m]

影响标志位 OV , Z , AC , C

SUBM A, [m] Subtract Data Memory from ACC with result in Data Memory

指令说明 将累加器中内容减去指定数据存储器的数据，把结果储存回数据存储器。如果结果为负，C 标志位清为 0，反之结果为正或 0，C 标志位设置为 1。

功能表示 [m] ← ACC – [m]

影响标志位 OV , Z , AC , C

SUB A, x Subtract immediate Data from ACC

指令说明 将累加器中内容减去立即数，把结果储存回累加器。如果结果为负，C 标志位清为 0，反之结果为正或 0，C 标志位设置为 1。

功能表示 ACC ← ACC – x

影响标志位 OV , Z , AC , C

SWAP [m] Swap nibbles of Data Memory

指令说明 将指定数据存储器的低 4 位与高 4 位互相交换。

功能表示 [m].3 ~ [m].0 ↔ [m].7 ~ [m].4

影响标志位 None

SWAPA [m] Swap nibbles of Data Memory with result in ACC

指令说明 将指定数据存储器的低 4 位与高 4 位互相交换，然后把结果储存回累加器且数据存储器的内容不变。

功能表示 ACC.3 ~ ACC.0 ← [m].7 ~ [m].4

ACC.7 ~ ACC.4 ← [m].3 ~ [m].0

影响标志位 None

SZ [m] Skip if Data Memory is 0

指令说明 如果指定数据存储器的内容为 0，则程序计数器再加 1 跳过下一条指令，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，程序继续执行下面的指令。

功能表示 Skip if [m] = 0

影响标志位 None

SZA [m] Skip if Data Memory is 0 with data movement to ACC

指令说明 将指定数据存储器的内容复制到累加器，如果值为 0，则程序计数器再加 1 跳过下一条指令，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，程序继续执行下面的指令。

功能表示 ACC ← [m]

Skip if [m] = 0

影响标志位 None

SZ [m].i Skip if bit i of Data Memory is 0

指令说明 如果指定数据存储器第 i 位为 0，则程序计数器再加 1 跳过下一条指令，由于取得下一指令时会要求插入一个虚指令，所以此指令为 2 个周期的指令。如果结果不为 0，程序继续执行下面的指令。

功能表示 Skip if [m].i = 0

影响标志位 None

TABRDC [m]　Read table (current page) to TBLH and Data Memory
指令说明　将表格指针 TBLP 所指的程序代码低字节(当前页)移至指定数据存储器且将高字节移至 TBLH。
功能表示　[m] ← 程序代码(低字节)
　　　　　TBLH ← 程序代码(高字节)
影响标志位　None

TABRDL [m]　Read table (last page) to TBLH and Data Memory
指令说明　将表格指针 TBLP 所指的程序代码低字节(最后一页)移至指定数据存储器且将高字节移至 TBLH。请注意 HT48R05A-1/HT48C05 并无此项功能。
指令说明　[m] ← 程序代码(低字节)
　　　　　TBLH ← 程序代码(高字节)
影响标志位　None

XOR A, [m]　Logical XOR Data Memory to ACC
指令说明　将存在累加器和指定数据存储器中的数据作 XOR 的运算，然后把结果储存回累加器。
功能表示　ACC ← ACC“XOR”[m]
影响标志位　Z

XORM A, [m]　Logical XOR ACC to Data Memory
指令说明　将存在指定数据存储器和累加器中的数据作 XOR 的运算，然后把结果储存回数据存储器。
功能表示　[m] ← ACC“XOR”[m]
影响标志位　Z

XOR A, x　Logical XOR immediate data to ACC
指令说明　将存在累加器中的数据和立即数作 XOR 的运算，然后把结果储存回累加器。
功能表示　ACC ← ACC“XOR”x
影响标志位　Z

第四章

汇编语言和编译器

4

源程序由汇编语言程序构成，由盛群编译器(Holtek Assembler)编译成目标文件(Object File)，再由连接器(Linker)连接并产生任务文件(Task File)。

源程序(source program)由语句(statement)和表格(look up table)组成，在编译器进行编译或程序执行时会给予指示，而语句是由助记符(mnemonic)、操作数(operand)和注解(comment)组成。

常用符号

下表描述了文章中所用到的常用符号

范　例	描　述
[*optional items*]	中括号内的项目是可选择的，下列的命令行语法中： **HASM** [*options*] *filename* [;] *options* 和分号都是可选的，而 *filename* 则必须设定。但是在指令操作数中的中括号则是指定存储器地址之用，必须要有。
{ *choice1* | *choice2*}	大括号和垂直线代表两个或更多的选项，大括号圈出这些选项而垂直线则用来分隔选项，只能有一个选项被选出。
Repeating elements…	三个连续的点表示允许输入更多同样形式的数据，例如以下的指令格式： **PUBLIC** *name1* [,*name2* [,…]] *name2* 之后的三个连续点表示允许输入更多的名称，只要每两个名称之间用逗号隔开即可。

语句语法

语句的语法格式如下：

[*名称*] [*操作项*] [*操作数项*] [*;注解*]

- 上述四个成员不一定都要指定。
- 每两个成员之间（除了注解）最少要以一个空格或一个 tab 符号分隔开。
- 成员的字型无大小写之分，换言之，编译器在编译之前会将小写字母改为大写字母。

名称

语句前可有标号以便于其它语句使用，如果名称当做标号使用，则必须在名称后紧接一个冒号(：)。名称由下列字符组成：

A～Z　a～z　0～9　?　_　@

在使用上有以下的限制：

- 不可使用数字 0～9 作为名称的第一个字符
- ? 不能单独作为名称
- 只有最前面的 31 个字符被认可

操作项

操作项定义两种形式的语句，伪指令与指令。伪指令用来指导编译器如何在编译时产生目标码。指令则是引导单片机执行各种运算。两者都会在编译时产生目标码，目标码会在执行时指导单片机的运作。

操作数项

操作数项定义伪指令与指令所使用的数据，由符号、常数、表达式和寄存器所组成。

注解

注解是对程序代码的一种叙述与说明。编译器不会编译它。任何在分号之后的文字均被视为注解。

编译伪指令

编译伪指令用来指导编译器如何在编译时产生目标码。编译伪指令可以依其行为细分如下。

条件编译伪指令

条件区段的格式如下：

IF
statements
［**ELSE**
statements］
ENDIF

→ 语法

IF *expression*
IFE *expression*

- 说明

  伪指令 **IF** 和 **IFE** 对其后的 *expression* 进行检测。
  如果 *expression* 的数值为真，换言之不为零，则在 **IF** 与 **ELSE** 或 **IF** 与 **ENDIF**（没有 **ELSE**）之间所有的语句会被编译。
  如果 *expression* 的数值为假，换言之为零，则在 **IFE** 与 **ELSE** 或 **IFE** 与 **ENDIF**（没有 **ELSE**）之间所有的语句会被编译。

- 范例

```
IF   debugcase
          ACC1     equ  5
          extern   username: byte
ENDIF
```

  在此范例中，如果符号 debugcase 的数值为真，也就是不为零，则变量 ACC1 的数值将被设定为 5，同时 username 被声明为外部变量。

→ 语法

IFDEF *name*
IFNDEF *name*

- 说明

  IFDEF 和 **IFNDEF** 的差异在检测 *name* 是否被定义。只要 *name* 已在前面定义为标号、变量或符号，则在 **IFDEF** 与 **ENDIF** 之间的语句都会被编译。相反的如果 *name* 还未被定义，则在 **IFNDEF** 与 **ENDIF** 之间的语句会被编译，条件编译伪指令提供最多 7 层的嵌套。

- 范例

```
IFDEF       buf_flag
            buffer  DB  20 dup (?)
ENDIF
```

  在此范例中，只要 buf_flag 被事先定义，即配置存储器空间给 buffer。

文件控制伪指令

→ 语法

INCLUDE *file-name*
或
INCLUDE "*file-name*"

- 说明

  此伪指令会在编译时，将包含进文件 *file-name* 的内容，嵌入至当前的源程序文件，并被视为源程序。编译器可提供最多 7 层的嵌套。

- 范例

```
INCLUDE     macro.def
```

  在此范例中，编译器将包含进文件 macro.def 内的源程序，嵌入至当前的源程序文件。

→ 语法

PAGE *size*

- 说明

  此伪指令指定程序列表文件(program listing file)中每一页的行数，其范围介于 10 行至 255 行之间，编译器的默认值为 60 行。

- 范例

```
PAGE 57
```

  在此范例中，程序列表文件的每一页最多为 57 行。

→ 语法

.LIST
.NOLIST

- 说明

  伪指令 **.LIST** 和 **.NOLIST** 用来决定是否要将源程序行存储到程序列表文件(program listing file)。**.NOLIST** 会禁止将其后的源程序存写到程序列表文件，而 **.LIST** 则会将其后的源程序存写到程序列表文件。编译器的默认值为 **.LIST**。

- 范例

```
.NOLIST
mov  a, 1
mov  b1, a
.LIST
```

  上面的范例中，被 **.NOLIST** 和 **.LIST** 所包围的两条指令将不会被写到程序列表文件。

→ 语法

.LISTMACRO
.NOLISTMACRO

- 说明

  伪指令**.LISTMACRO** 会引导编译器将宏指令中包括注解的所有语句都存写到程序列表文件。伪指令**.NOLISTMACRO** 则中止写入所有宏指令的语句。编译器的默认值为**.NOLISTMACRO**。

→ 语法

.LISTINCLUDE
.NOLISTINCLUDE

- 说明

  .LISTINCLUDE 会引导编译器将所有包含文件(included files)的内容写入程序列表文件中，**.NOLISTINCLUDE** 则会禁止编译器将其后的包含文件的内容写进程序列表文件，编译器默认值为**.NOLISTINCLUDE**。

→ 语法

MESSAGE '*text-string*'

- 说明

  伪指令 **MESSAGE** 引导编译器将 *text-string* 显示于屏幕上，'*text-string*'的字符必须使用一对单引号括起来。

→ 语法

ERRMESSAGE '*error-string*'

- 说明

  伪指令 **ERRMESSAGE** 引导编译器显示错误信息，'*error-string*'的字符必须使用一对单引号括起来。

程序伪指令

→ 语法（注解）

; text

- 说明

  注解是以分号(semicolon)开始的字符所组成，而由回车/换行符结束。

→ 语法

name **.SECTION** [*align*] [*combine*] '*class*'

- 说明

  伪指令**.SECTION** 标明程序段(program section)或数据段(data section)的起始地址。程序段是由指令和/或数据所组成，这些指令与数据的地址是以该程序段的段名 *name* 为起始标准而定出的。程序段的段名 *name* 可以是唯一的或者是与程序中其它段的段名相同，若两个程序段设定有完全相同的名称(complete name)，则被视为是同一个程序段。

  选项 *align* 定义程序段起始地址的类型，可以用下列中的一种：
  BYTE 以任意字节地址(byte)当做起始地址（编译器的默认形式）
  WORD 以字地址（word，两个字节，即偶数地址）当做起始地址
  PARA 以节段地址（paragraph，16 的倍数）当做起始地址
  PAGE 以分页地址（page，256 的倍数）当做起始地址
  针对 CODE 类别(class)的程序段，是以一条指令当做一个字节地址。**BYTE** 会将程序段的起始地址安排在任何指令的地址，**WORD** 则将程序段的起始地址安排在偶数的指令地址，**PARA** 将程序段的起始地址安排在 16 倍数的指令地址，而 **PAGE** 则将程序段的起始地址安排在 256 倍数的指令地址。对于 DATA 类别的数据段而言，是以一个字节（8 位/字节）当做地址的计算单位。**BYTE** 会将数据段的起始地址安排在任何字节地址，**WORD** 则将数据段的起始地址安排在偶数地址，**PARA** 将数据段的起始地址安排在 16 倍数的地址，而 **PAGE** 会将数据段的起始地址安排在 256 倍数的地址。

选项 *combine* 定义如何结合有完全相同名称的程序段的方法，可以选用下列中的一种：

– COMMON
  将具有完全相同名称的所有程序段的起始地址安排在同一个地址，所使用的存储器长度则是以最长的程序段的长度为准。

– AT *address*
  此选项是指定程序段的起始地址为 address，一个固定地址。编译器及连接器不能把它安排到其它的地址，而其内的标号(label)和变量(variable)的地址可以直接从 *address* 计算出。除了不可有前置引用(forward reference)的变量或符号外，可以使用任何合乎规定的表达式来表示 address。而运算结果的数值必须是合法的 ROM/RAM 存储器地址，且不能超出 ROM/RAM 的大小范围。

如果没有设定 *combine* 的形式，则该程序段是可结合的，换句话说，此程序段和其它具有完全相同名称的程序段可以连接成一个单一的程序段。

Class 是定义段存放的存储器类别。相同类别的段被安排在存储器中的连续区域。以其输入的先后顺序一个个紧接地安排在存储器中。类别名称为 **CODE** 的程序段将会放置在程序存储器(program memory - ROM)，而类别名称为 **DATA**的数据段则是存储在数据存储器(data memory - RAM)。完整的段伪指令包括一个 section 名称和一个 class 名称。在此伪指令之后，直到下一段伪指令之前的所有指令及数据，都属于此段。

→ 语法
ROMBANK *banknum section-name* [,*section-name*,...]

• 说明
  此伪指令是用来声明程序存储器(program memory)的某一区块(bank)所包含的程序段。*banknum* 指定程序存储器的区块编号，范围从 0 到单片机的最大程序存储器区块数。*section-name* 则是先前已定义的程序段的名称。可以在同一个存储器区块内声明多个程序段，只要这些被声明的程序段的总和不超过 8K 字。如果程序中没有声明此伪指令，则所有类别为 CODE 的程序段都被视为属于区块 0 (bank 0)。如果某个类别为 CODE 的程序段没有被声明属于任何程序存储器的区块内，此程序段将被视为属于区块 0。

→ 语法

RAMBANK *banknum section-name* [,*section-name*,...]

- 说明

  此伪指令与 **ROMBANK** 相似，不同的地方是声明数据存储器(data memory)的区块所包含的数据段(data section)。数据存储器区块的大小是 256 字节。

→ 语法

END

- 说明

  此伪指令声明程序结束，因此应该避免在任何包含文件(included file)中加入此伪指令。

→ 语法

ORG *expression*

- 说明

  此伪指令会将 *expression* 的计算数值设定给编译器的地址计数器(location counter)，其后的程序代码和数据偏移地址将根据 *expression* 所计算的偏移量做相对的调整。程序代码和数据偏移量与伪指令 **ORG** 所在的程序段的起始地址有关，程序段的属性会决定偏移量的实际值(是绝对地址或相对地址)。

- 范例

```
ORG 8
mov A, 1
```

  在此范例中，语句 `mov A, 1` 的地址是在当前程序段的第 8 个地址。

→ 语法

PUBLIC *name1* [,*name2* [,...]]
EXTERN *name1*:*type* [,*name2*:*type* [,...]]

- 说明

  伪指令 **PUBLIC** 声明变量或标号可被其它程序文件中的程序模块使用，也就是公用变量或标号。另一方面，伪指令 **EXTERN** 则声明程序要使用的外部变量、标号或符号的名称和类型。这里提到的类型可使用下列四种形式中的一种：**BYTE**、**WORD**、**BIT**（这三种形式适用于数据变量）和 **NEAR**（用于调用或跳转的标号形式）。

- 范例

```
PUBLIC  start, setflag
EXTERN  tmpbuf:byte
CODE        .SECTION 'CODE'
start:
     mov    a, 55h
     call   setflag
     ...
setflag     proc
     mov    tmpbuf, a
     ret
setflag     endp
end
```

  在此范例中，标号 start 和程序 setflag 都被声明是公用变量，而其它源程序文件中的程序可以使用这些变量。变量 tmpbuf 则被声明是外部变量。在其它的源程序文件中，一定有一个名为 tmpbuf 的 byte 型变量定义，而且被声明为公用变量。

→ 语法

name **PROC**
name **ENDP**

- 说明

  伪指令 **PROC** 和 **ENDP** 用来定义一段可被其它程序调用或跳转到的程序代码。必须要指定一个名称 *name* 给 **PROC** 代表此程序(procedure)第一条指令的地址，而编译器会将标号的值设定至地址计数器中。

- 范例

```
toggle      PROC
mov         tmpbuf, a
mov         a, 1
xorm        a, flag
mov         a, tmpbuf
ret
toggle      ENDP
```

→ 语法

`[label:] DC expression1 [,expression2 [,...]]`

- 说明

  伪指令 **DC** 会将 *expression1* 及 *expression2* 等的值存储在存储器的连续位置。此伪指令只能使用于 CODE 类别的程序段之内。*expression1* 及 *expression2* 的数值将视单片机的程序存储器的宽度大小而定，编译器会将任何多余的位清除掉，*expression1* 必须为数值或标号。此伪指令通常被用在程序段之内建立表格以便查询。

- 范例

  ```
  table: DC 0128H, 025CH
  ```

  在此范例中，编译器会预留两个地址的 ROM 空间，并将 0128H 和 025CH 储存至这两个地址中。

数据定义伪指令

汇编程序由语句和注解组成。语句或注解则是由字符、数字和名称构成。汇编语言支持整数数字。整数可由二进制、八进制、十进制或十六进制加以表示（配合字尾的基数），如果未选基数，则编译器会使用默认值(十进制)，下表为可用的基数。

基 数	型 态	数 字
B	二进制	01
O	八进制	01234567
D	十进制	0123456789
H	十六进制	0123456789ABCDEF

→ 语法

```
[name]  DB  value1 [,value2 [,...]]
[name]  DW  value1 [,value2 [,...]]
[name]  DBIT
[name]  DB  repeated-count DUP(?)
[name]  DW  repeated-count DUP(?)
```

- 说明

  上述伪指令会引导编译器在数据存储器(data memory)内保留空间给变量 *name* (如果有指定 *name*)。存储器保留的空间大小则由其后的个数及数据类型，或由重复次数及数据类型来决定。由于单片机的数据存储器无法事先记录数据内容，编译器不会对数据存储器做初始值的设定，因此 *value1* 和 *value2* 必须为“?”，表示只是保留存储器空间给程序执行时使用，并没有设定其初始值。**DBIT** 只保留一个位，编译器会将每 8 个 **DBIT** 整合在一起并且保留一字节给这 8 个 **DBIT** 变量。

- 范例

```
DATA        .SECTION     'DATA'
tbuf        DB  ?
chksum      DW  ?
flag1       DBIT
sbuf        DB  ?
cflag       DBIT
```

  在此范例中，编译器保留字节地址 0 给变量 tbuf、字节地址 1 和 2 给变量 chksum、字节地址 3 的第 0 位给变量 flag1、字节地址 4 给变量 sbuf 以及字节地址 3 的第 1 位给变量 cflag。

→ 语法

name **LABEL {BIT | BYTE | WORD}**

- 说明

  此伪指令会将 *name* 的地址设定为与其后的变量相同的存储器地址。
- 范例

```
lab1        LABEL        WORD
d1          DB  ?
d2          DB  ?
```

  在这个范例中，d1 是 lab1 的低字节，而 d2 则是 lab1 的高字节。

→ 语法

name **EQU** *expression*

- 说明

  EQU 伪指令将 *expression* 传送给 *name*，从而产生一个绝对符号、别名或文字符号。绝对符号是一个代表 16 位值的名称；别名则替代另一个符号；而文字符号则是代表一串字符组合的名称。*name* 必须是唯一的，即先前未被定义过。*expression* 可以是一个整数、字符串常数、指令助记符、数字表达式或地址表达式。
- 范例

```
accreg      EQU   5
bmove       EQU   mov
```

  在这个范例中，变量 accreg 等于 5，而 bmove 相当于指令 mov。

宏指令

宏指令定义一个名称来代表一段源程序语句，而在源程序文件中可以重复用这个名称以取代这段语句。在编译时，编译器会自动将每一个宏指令的名称用宏指令所定义的程序语句来取代。

宏指令可以在源文件的任何地方被定义，只要调用此宏指令是在宏指令定义之后即可。宏指令的定义中，可以调用先前已经被定义的其它宏指令，如此将形成一种嵌套的结构，编译器提供最多 7 层的嵌套。

→ 语法

```
name MACRO  [dummy-parameter [,...]]
     statements
     ENDM
```

在宏指令中，可以使用伪指令 **LOCAL** 来定义只能在宏指令本体内使用的变量。

→ 语法

```
name  LOCAL  dummy-name [,...]
```

- 说明

  宏指令 **LOCAL** 用来定义只能在宏指令本体内使用的符号，使用时必须定义在 **MACRO** 伪指令之后的第一行。*dummy-name* 是一个暂时使用的名称，当宏指令被调用展开时，它将被一个唯一的名称所取代。编译器会对 *dummy-name* 产生对应的实际名称。这个实际名称的格式为`??digit`，其中 digit 数字为十六进制且范围由 0000 至 FFFF。当 **MACRO/ENDM** 的定义区段中使用到一些标号(label)时，要将这些标号加入 **LOCAL** 伪指令中，否则当 **MACRO** 被源文件多次引用时，相同的标号名称会重复出现在程序中，编译器会发布程序错误的信息。

  下面的范例中，`tmp1` 和 `tmp2` 都是形式参数，当调用此宏指令时，都会被实际参数所取代，`label1` 和 `label2` 都被声明为 **LOCAL**，如果没有其它的 **MACRO** 被引用，在第一次引用时将分别被`??0000` 和`??0001` 所取代，如果没有声明 **LOCAL**，`label1` 和 `label2` 则会类似于源程序中的标号声明，而在第二次调用此宏指令时，就会出现重复定义的错误信息。

```
Delay  MACRO   tmp1, tmp2
     LOCAL     label1, label2
     mov       a, 70h
     mov       tmp1, a
label1:
     mov       tmp2, a
label2:
     clr       wdt1
     clr       wdt2
     sdz       tmp2
     jmp       label2
     sdz       tmp1
     jmp       label1
     ENDM
```

下面的源程序将会调用名为 Delay 宏指令

```
; T.ASM
; Sample program for MACRO.
.ListMacro
Delay MACRO  tmp1, tmp2
      LOCAL  label1, label2
      mov    a, 70h
      mov    tmp1, a
label1:
      mov    tmp2, a
label2:
      clr    wdt1
      clr    wdt2
      sdz    tmp2
      jmp    label2
      sdz    tmp1
      jmp    label1
      ENDM

data .section 'data'
BCnt db ?
SCnt db ?

code .section at 0 'code'
Delay BCnt, SCnt
end
```

编译器会将宏指令 Delay 展开如下列的程序。请注意在宏指令本体内的第 4 行到第 17 行，它们的偏移地址(offset)都是 0000，也就是宏指令在定义时，本体内的指令并不占用存储器空间。在程序第 24 行调用 Delay 宏指令时，它就被展开成 11 行，也就是展开为宏指令程序。形式参数 tmp1 和 tmp2 分别被实际参数 BCnt 和 SCnt 所取代。

```
File: T.asm          Holtek Cross-Assembler   Version 2.80      Page 1
 1  0000                     ; T.ASM
 2  0000                     ; Sample Program for MACRO.
 3  0000                     .ListMacro
 4  0000                     Delay MACRO  tmp1, tmp2
 5  0000                           LOCAL  label1, label2
 6  0000                           mov    a, 70h
 7  0000                           mov    tmp1, a
 8  0000                     label1:
 9  0000                           mov    tmp2, a
10  0000                     label2:
11  0000                           clr    wdt1
12  0000                           clr    wdt2
13  0000                           sdz    tmp2
14  0000                           jmp    label2
15  0000                           sdz    tmp1
16  0000                           jmp    label1
17  0000                           ENDM
18  0000
19  0000                     data .section 'data'
20  0000  00                 BCnt db ?
21  0001  00                 SCnt db ?
22  0002
23  0000                     code .section at 0 'code'
24  0000                     Delay BCnt, SCnt
24  0000  0F70          1          mov    a, 70h
24  0001  0080         R1          mov    BCnt, a
24  0002                1     ??0000:
24  0002  0080         R1          mov    SCnt, a
24  0003                1     ??0001:
24  0003  0001          1          clr    wdt1
24  0004  0005          1          clr    wdt2
24  0005  1780         R1          sdz    SCnt
24  0006  2803          1          jmp    ??0001
24  0007  1780         R1          sdz    BCnt
24  0008  2802          1          jmp    ??0000
25  0009                     end

      0 Errors
```

汇编语言指令

指令的语法形式如下：

[*name*:] *mnemonic* [*operand1* [,*operand2*]] [; *comment*]

其中

name:	→	符号名称
mnemonic	→	指令名称（关键字）
operand1	→	寄存器 存储器地址
operand2	→	寄存器 存储器地址 立即数

名称

名称是由字母、数字或特殊字符所组成的，可以当标号(label)使用。当标号使用时，必须在名称后面紧接一个冒号(colon)。

助记符

助记符是源程序中使用的指令名称，它取决于源程序所使用的单片机型号。

操作数、运算子和表达式

操作数（源操作数或目的操作数）定义被指令所使用的数值。它们可以是常数、变量、寄存器、表达式或关键字。当使用指令时，必须谨慎选择正确的操作数，即源操作数和目的操作数。符号$是一个特殊的操作数，它代表当前的地址。

表达式是由操作数所组成，在程序编译时用来计算出数值或存储器地址。表达式是常数、符号以及任何被算术运算子分隔的常数和符号组合。

运算子定义表达式中各操作数之间的运算动作。编译器提供了许多运算子去处理操作数。有些运算子只处理常数，有些则处理存储器数值，也有两者兼具的。运算子在程序编译时就会直接计算出数值。以下是编译器所提供的运算子。

- 算术运算子　+ – * / %（MOD）
- SHL 和 SHR 运算子
  - 语法

```
expression  SHR  count
expression  SHL  count
```

这些位平移运算子的值全都为常数。*expression* 依照 *count* 所指定的数目向右移 **(SHR)** 或向左移 **(SHL)**，如果被平移的位超过有效位数时，则对应的位会以 0 填满，如：

```
mov  A, 01110111b  SHR 3  ; result  ACC=00001110b
mov  A, 01110111b  SHL 4  ; result  ACC=01110000b
```

- 逻辑运算子 NOT、AND、OR、XOR
  - 语法

    NOT *expression*
    expression1 **AND** *expression2*
    expression1 **OR** *expression2*
    expression1 **XOR** *expression2*

    NOT 各位的 1 阶补码

    AND 各位 AND 运算

    OR 各位 OR 运算

    XOR 各位 XOR 运算

- OFFSET 运算子
  - 语法

    OFFSET *expression*

  OFFSET 运算子返回 *expression* 的偏移地址。*expression* 可以是标号、变量或其它直接存储器的操作数。被 **OFFSET** 运算子所返回的数值必须是立即数。

- LOW、MID 和 HIGH 运算子
  - 语法

    LOW *expression*
    MID *expression*
    HIGH *expression*

  如果 *expression* 的结果为一个立即数的话，则 **LOW/MID/HIGH** 运算子返回值就是 *expression* 的值，而且是分别取此数值的低/中/高字节。但是如果 *expression* 是标号，则 **LOW/MID/HIGH** 运算子将取得此标号所在的程序存储器地址的低/中/高字节的数值。

- BANK 储存区块运算子
  - 语法

    BANK *name*

  BANK 运算子会返回程序段所在的存储器区块的编号，此程序段的名称是 *name*。如果 *name* 是标号，则返回 ROM 程序存储器区块。如果 *name* 是数据变量则返回 RAM 数据存储器区块。存储器区块的数值格式与寄存器 BP 的格式相同，请参考各单片机的规格。（注意：不同的单片机可能有不同的 BP 格式。）

  范例 1:

```
mov  A, BANK start
mov  BP, A
jmp  start
```

  范例 2：

```
mov  A, BANK var
mov  BP, A
mov  A, OFFSET var
mov  MP1, A
mov  A, R1
```

- 运算子的优先权

优先权	运算子
1(Highest)	( ), [ ]
2	+ , – (unary), LOW, MID, HIGH, OFFSET, BANK
3	*, /, %, SHL, SHR
4	+, – (binary)
5	> (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to)
6	= = (equal to), != (not equal to)
7	! (bitwise NOT)
8	& (bitwise AND)
9(Lowest)	| (bitwise OR), ^ (bitwise XOR)

其它

前置引用

当标号、变量名称和其它符号在源代码中被声明之前，编译器允许它们被使用(前置命名引用)，但是在伪指令 **EQU** 右边的符号是不允许前置引用的。

局部标号

局部标号有固定的形式$number。其中 number 可以为 0 至 29，局部标号除了可以重复使用外，其它功用与一般标号相同。局部标号必须使用在任意两个连续的标号之间而且同样的局部标号名称也可以用在其它的两个连续标号之间。在编译源程序文件之前，编译器会将每一个局部标号转换成唯一的标号。任何两个连续标号之间，最多可以定义 30 个局部标号。

范例

```
Label1:                         ; label1
          $1:                   ;; local label
                mov a, 1
                jmp $3
          $2:                   ;; local label
                mov a, 2
                jmp $1
          $3:                   ;; local label
                jmp $2
Label2:                         ; label
                Jmp $1
          $0:                   ;; local label
                jmp Label1
          $1:   jmp $0
Label3:
```

汇编语言保留字

下表是汇编语言上使用的保留字。

- 保留字（伪指令、运算子）

$	DUP	INCLUDE	NOT
*	DW	LABEL	OFFSET
+	ELSE	.LIST	OR
–	END	.LISTINCLUDE	ORG
.	ENDIF	.LISTMACRO	PAGE
/	ENDM	LOCAL	PARA
=	ENDP	LOW	PROC
?	EQU	MACRO	PUBLIC
[ ]	ERRMESSAGE	MESSAGE	RAMBANK
AND	EXTERN	MID	ROMBANK
BANK	HIGH	MOD	.SECTION
BYTE	IF	NEAR	SHL
DB	IFDEF	.NOLIST	SHR
DBIT	IFE	.NOLISTINCLUDE	WORD
DC	IFNDEF	.NOLISTMACRO	XOR

- 保留字（指令助记符）

ADC	HALT	RLCA	SUB
ADCM	INC	RR	SUBM
ADD	INCA	RRA	SWAP
ADDM	JMP	RRC	SWAPA
AND	MOV	RRCA	SZ
ANDM	NOP	SBC	SZA
CALL	OR	SBCM	TABRDC
CLR	ORM	SDZ	TABRDL
CPL	RET	SDZA	XOR
CPLA	RETI	SET	XORM
DAA	RL	SIZ	
DEC	RLA	SIZA	
DECA	RLC	SNZ	

- 保留字（寄存器名称）

A	WDT	WDT1	WDT2

编译器选项

编译器选项可以通过 HT-IDE3000 中的 Options 菜单的 Project 命令来设定，编译器的选项位于 Project Option 对话框的中心部分。

可在符号定义(Define Symbol)编辑框中定义符号。

→ 语法

symbol1 [=value1] [,symbol2 [=value2] [,...]]

- 范例

```
debugflag=1, newver=3
```

产生列表文件的检查框可用来指定是否要生成列表文件(Listing file)，如果检查框被选中，则要生成列表文件，否则将不会产生列表文件。

编译列表文件格式

编译列表文件包含源程序的列表和概要信息。每页的第一行是标题，内容则包括公司名称、编译器版本、源文件名称、编译的日期、时间以及页码。

源程序列表

在源程序中的每行语句都以下列的格式输出到编译列表文件：

[line-number] offset [code] statement

- *Line-number* 是指语句在源程序文件的第几行，从一个语句开始计算起(4 位十进制数)。
- *offset*-是从语句所在的程序段开始到这个语句的存储器地址的偏移量(4 位十六进制数)。
- *code*-只有会产生机器码(machine code)或数据的语句才会出现此项(两个 4 位十六进制数)。

如果数值在编译时已确定的话，会用十六进制数字表示 *code* 的数值，否则的话，将使用适当的标志位表明应该使用何种方式去计算此数值。下列两个标志位可能会出现于 *code* 项之后。

R → 需要重新安置地址(连接器解决此状况)

E → 需要参考外部符号(连接器解决此状况)

下列标志位可能会出现于 *code* 项之前。

= → **EQU** 或等号

code 项中可能出现下列的符号或数字。

---- → 代表程序段的起始地址(连接器会解决此符号)

nn[xx]→ **DUP** 符号：nn **DUP**(?)重复次数

- *statement*-源文件对应的源程序语句或是宏指令所展开的语句，在语句之前可能会出现下列的符号。

  n → 宏指令展开时的嵌套层次

  C → 此语句是从包含文件(**INCLUDE** 文件)引进的

- 总结

```
0         1         2         3         4         5         6
1234567890123456789012345678901234567890123456789012345678901234567890...
llll  oooo hhhh hhhh EC source-program-statement
                     Rn
```

IIII → 行号（4 位数，向右靠齐）

oooo → 代码的偏移量（4 位数）

hhhh → 两个 4 位数机器码

E → 外部引用

C → 从包含文件加入的语句

R → 需要重新安置地址

n → 宏指令展开后的嵌套层次

编译总结

在编译列表文件的结尾处会统计此次编译所发生的警告及错误的总数。

其它

在编译期间如果发生错误，则错误信息和编号会直接出现在发生错误的语句下方。

→　编译列表文件的范例

```
File: SAMPLE.ASM    Holtek Cross-Assembler  Version 2.86      Page 1

   1  0000                  page 60
   2  0000                  message    'Sample Program 1'
   3  0000
   4  0000                  .listinclude
   5  0000                  .listmacro
   6  0000
   7  0000                  #include "sample.inc"

   1  0000                C pa     equ    [12h]
   2  0000                C pac    equ    [13h]
   3  0000                C pb     equ    [14h]
   4  0000                C pbc    equ    [15h]
   5  0000                C pc     equ    [16h]
   6  0000                C pcc    equ    [17h]
   7  0000                C

   8  0000
   9  0000                  extern extlab : near
  10  0000                  extern extb1 : byte
  11  0000
  12  0000                  clrpb macro
  13  0000                  clr pb
  14  0000                  endm
  15  0000
  16  0000                  clrpa macro
  17  0000                  mov a, 00h
  18  0000                  mov pa, a
  19  0000                  clrpb
  20  0000                  endm
  21  0000
  22  0000                  data .section 'data'
  23  0000  00              b1    db ?
  24  0001  00              b2    db ?
  25  0002  00              bit1  bit
  26  0003
  27  0000                  code .section 'code'
  28  0000  0F55            mov  a, 055h
  29  0001  0080       R    mov  b1, a
  30  0002  0080       E    mov  extb1, a
  31  0003  0FAA            mov  a, 0aah
  32  0004  0093            mov  pac, a
  33  0005                  clrpa
  33  0005  0F00        1 mov  a, 00h
  33  0006  0092        1 mov  [12h], a
  33  0007              1 clrpb
  33  0007  1F14        2 clr  [14h]
  34  0008  0700        R  mov  a, b1
  35  0009  0F00        E  mov  a, bank extlab
  36  000A  0F00        E  mov  a, offset extb1
  37  000B  2800        E  jmp  extlab
  38  000C
  39  000C  1234 5678     dw 1234h, 5678h, 0abcdh, 0ef12h
            ABCD EF12
  40  0010                 end

        0 Errors
```

第三部份

开发工具

第五章

单片机开发工具

5

为简化应用程序的开发过程，支持工具的重要性和有效性对于单片机来说是不可低估的。为了支持所有系列的 MCU，盛群用心的提供了具有完整功能的工具，让用户在开发与使用上更加便利，例如众所周知的 HT-IDE 集成开发环境，软件方面有 HT-IDE3000 软件，提供友好的视窗接口以便进行程序的编辑及除错，同时硬件方面为 HT-ICE 仿真器，提供多种实时仿真功能，包含多功能跟踪、单步执行和设定断点功能。HT-IDE 开发系统提供完整的接口卡与定期软件服务包的更新，因此保证设计者可以有最佳的工具，且能以最高效率进行单片机应用程序的设计与开发。

HT-IDE 集成开发环境

HT-IDE(Holtek Integrated Development Environment)是一个具有高效能，使用于设计盛群 8 位 MCU 应用程序的集成开发环境。系统中的硬件及软件工具能帮助客户使用盛群 8 位 MCU 芯片，快速且容易的开发应用程序。在 HT-IDE 中最主要的组件为 HT-ICE，它提供盛群 8 位单片机的实时仿真功能，以及强有力的除错和跟踪功能。最近版本的 HT-ICE 仿真器更进一步集成 OTP writer 烧录器在仿真器上，提供使用者从程序设计、除错到烧录所有的功能。

在软件方面，HT-IDE3000 开发系统提供友好的工作平台。此平台将所有的软件工具，例如编辑器、编译器、连接器、函数库管理器和符号除错器，并入到视窗环境，使程序开发过程更为容易。HT-IDE3000 还提供软件仿真功能，无需接上 HT-ICE 仿真器，就可以进行程序开发。此软件仿真器可以仿真盛群 8 位 MCU，以及 HT-ICE 硬件的所有基本功能。

HT-IDE3000 使用手册中包含 HT-IDE 开发系统的相关细节。为了确保开发系统包含有最新的单片机和软件更新信息，盛群也定期提供 HT-IDE3000 服务软件(Service Pack)以配合 HT-IDE3000 的安装。这些服务软件不是用来取代 HT-IDE3000，它必须要在 HT-IDE3000 系统软件安装后才被安装。

HT-IDE3000 开发系统具有下列的特性：

→ 仿真

- 程序指令的实时仿真

→ 硬件

- 使用及安装容易
- 可使用内部或外部振荡器
- 断点功能
- 支持跟踪功能与触发能力的跟踪仿真器
- HT-ICE 通过打印口与计算机连接
- 使用者的应用电路板通过 I/O 接口卡连接至 HT-ICE
- HT-ICE 中集成 OTP 烧录器

→ 软件

- 通用的视窗软件
- 源程序层次的除错方式(符号除错器)
- 支持多个源程序文件的工作平台(一个应用项目可以包含一个以上的源程序文件)
- 所有的工具都用于开发、除错、评估和产生最后的应用程序代码(Mask ROM file)
- 可以建立公用程序的函数库，之后被连接到另一个项目去使用
- 软件仿真器不需要连接 HT-ICE 硬件即可进行程序的仿真和除错
- 虚拟外围器件管理(VPM)可仿真外围器件的行为
- LCD 仿真器可仿真 LCD 面板的动作

盛群单片机仿真器(HT-ICE)

对于盛群的 8 位单片机而言，盛群的 ICE 是全功能的仿真器，系统中的硬件及软件工具能帮助客户快速方便的开发应用程序。系统中最主要的是硬件仿真器，除了能够有效地提供除错和跟踪功能之外，还能以实时的方式进行盛群 8 位 MCU 的仿真工作。在软件方面，HT-IDE3000 开发系统提供友好的工作平台，将所有软件工具，例如编辑器、编译器、连接器、函数库管理器和符号除错器，合并到视窗环境。此外系统在软件仿真模式下不需连接 HT-ICE 硬件即能执行程序的仿真。

HT-ICE 接口卡

HT-ICE 的接口卡可以被许多的应用电路使用，但是使用者也可自行设计接口卡，将必要的接口电路放在他们自己的接口卡上，使用者可以直接把他们的应用电路板连接到 HI-ICE 的 CN1 和 CN2 连接器。

OTP 烧录器

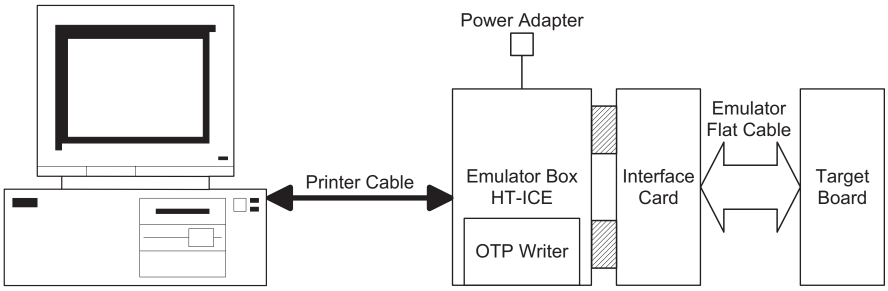
所有盛群的 OTP 芯片都有烧录器支持。对于工业级的 OTP 芯片烧录而言，盛群 OTP writer 工具提供一个快速且有效的方法，来进行 OTP 的小规模量产烧录。最新版本的 HT-ICE 仿真器更进一步的将 OTP writer 集成在 HT-ICE 仿真器上，提供使用者从程序设计、除错到烧录验证的所有必需功能。另外有更多的烧录器供应商可提供有效及更大容量的烧录服务。请参阅网站以获得进一步供应商情况。

OTP 适配卡

OTP 烧录器本身提供一个标准的芯片插座，而 OTP 适配卡则是使用在烧录其它封装形式的 OTP 芯片，这些封装形式的芯片无法在标准芯片插座上烧录，需要在 OTP 烧录器插上此适配卡才能烧录。

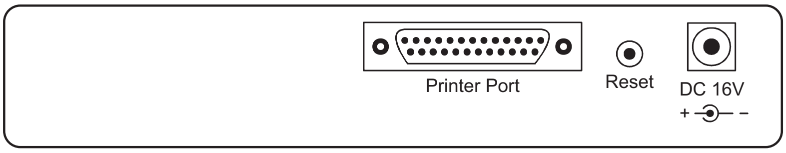
系统配置

HT-IDE 集成开发环境硬件配置如下，主机需为 Pentium 兼容机器(Windows 95/98/NT/2000/XP 或更新)。注意在 Windows NT/2000/XP 系统下安装 HT-IDE3000 时，需在 Supervisor Privilege 模式下执行 HT-IDE3000 软件安装。

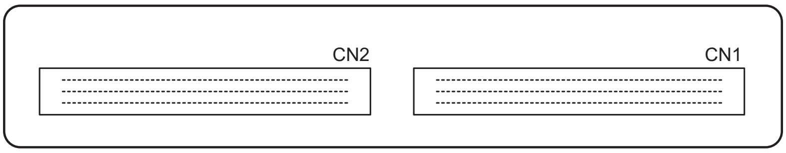


HT-IDE 集成开发环境包含下列硬件组成：

- HT-ICE 机台，包含印刷电路板 PCB，其中有一条打印口用于连接硬件仿真器与主机、I/O 接口连接头以及电源指示灯 LED，如下图所示。
- 连接目标电路板与 HT-ICE 机台的 I/O 接口卡
- 变压器(输出 16V)
- 25 脚的 D 型打印并口线
- OTP writer 烧录器



HT-ICE Rear View



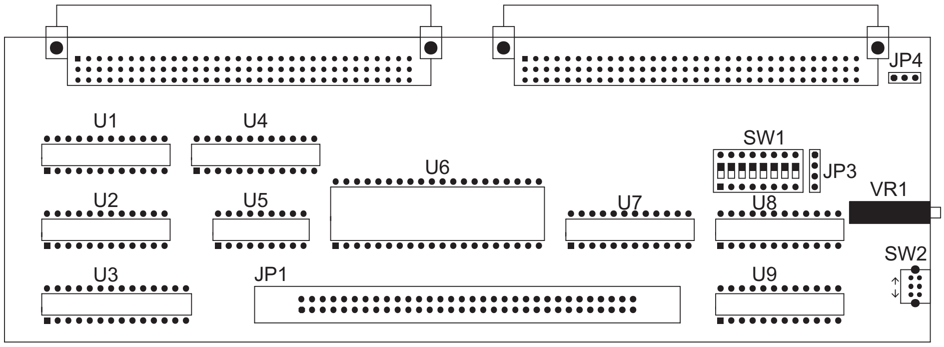
HT-ICE Front View

HT-ICE 接口卡设置

I/O 接口卡(TPCB48MIO-A-1)连接应用电路板与 HT-ICE，它提供下列功能：

- 外部时钟
- 外部输入的跟踪信号
- MCU 引脚配置

外部的时钟来源有两种模式：RC 和 Crystal，使用 Crystal 模式时，必须将 JP4 位置的第二和第三脚短路，然后在 Y1 插入合适的晶振。而 RC 模式则是将 JP4 位置的第一和第二脚短路，且可经 VR1 来调整系统频率。参考 HT-IDE3000 使用者手册中 Tools 菜单中的 Mask Option 命令选择时钟来源和系统频率。



有四个外部信号跟踪输入，地址在 JP3 上被标示为 ET0~ET3，作为内部断点触发或者跟踪功能使用。想得到更多相关的信息，请参考 HT-IDE3000 使用者手册中断和应用程序跟踪的章节。

开关 SW1 需要配合单片机型号以及下表设定使用：

型号	封装	SW1							
		1	2	3	4	5	6	7	8
HT48R05A-1 HT48R06A-1	18DIP	OFF	OFF	ON	OFF	ON	OFF	OFF	——

开关 SW2 需设定如下位置：

SW2	功能
↑	HT48XA6 carrier output
↓	其他单片机

U5 的引脚配置符合 18-pin DIP 封装，I/O 接口卡的 VME 连接器直接连接至 HT-ICE 的 CN1 和 CN2 插口。

安装

系统需求

安装 HT-IDE3000 系统的硬件及软件需求如下：

- Pentium 等级以上 CPU 之 PC/AT 兼容机器
- SVGA 彩色显示器
- 至少 32M 以上的 RAM
- CD ROM 装置(CD 安装者需要)
- 至少 20M 以上的硬盘空间
- 具有并行口，可连接 PC 和 HT-ICE
- 操作系统 Windows 95/98/NT/2000/XP

* Window 95/98/NT/2000/XP 是 Microsoft 公司注册商标

硬件安装

- 步骤 1
  将电源变压器插入 HT-ICE 的电源插孔
- 步骤 2
  通过 I/O 接口卡或排线连接目标电路板至 HT-ICE
- 步骤 3
  使用打印并口线连接 HT-ICE 至主机

此时 HT-ICE 上的 LED 应该是亮的，如果不是，则重新操作连接的步骤或与代理商联系。

警告: 请小心使用电源变压器，勿使用输出不是 16V 的变压器，否则可能导致 HT-ICE 损坏，因此强烈建议使用由盛群所提供的变压器。首先将电源变压器插入 HT-ICE 的电源插座。

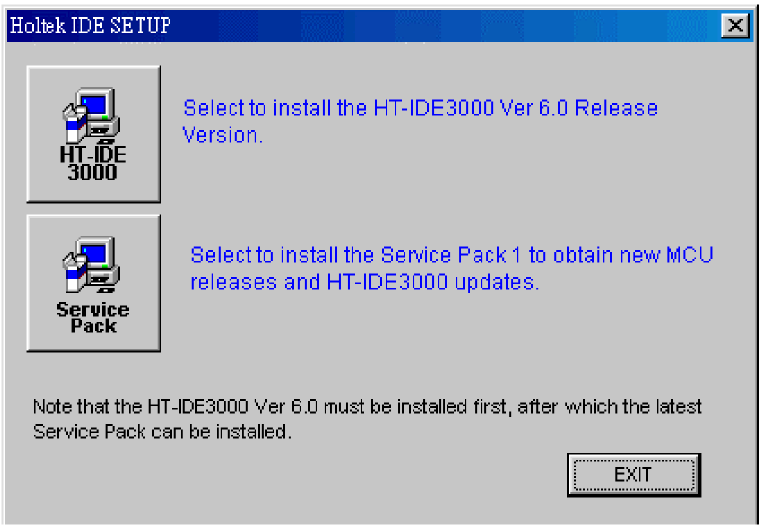
软件安装

- 步骤 1

  将 HT-IDE3000 CD 放入 CD ROM 装置中，将出现下列的对话框。



按下<HT-IDE3000>按钮，下列的对话框会出现。

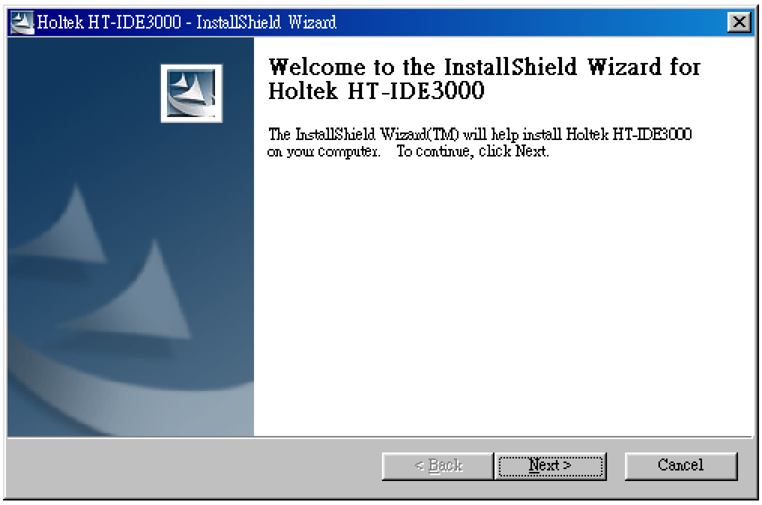


依照你想要安装的功能，请按下<HT-IDE3000>或<Service Pack>按钮。如果是首次安装，请按<HT-IDE3000>，如果已安装 HT-IDE3000，而需要更新版本时，请按<Service Pack>。

以下为选择安装<HT-IDE3000>的范例说明，按下<HT-IDE3000>。

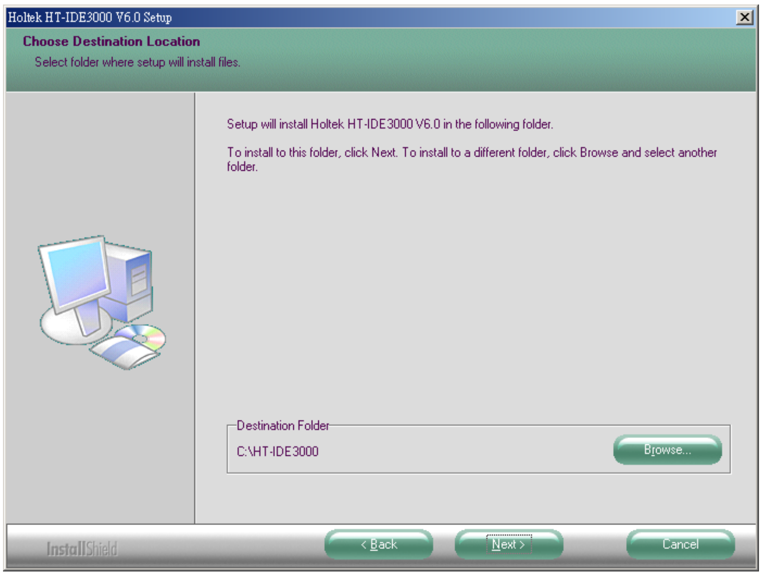
- 步骤 2

按下<Next>按钮(继续安装)或按下<Cancel>按钮(中止安装)。



- 步骤 3

下面显示的对话框会要求使用者输入安装处的文件夹名称。

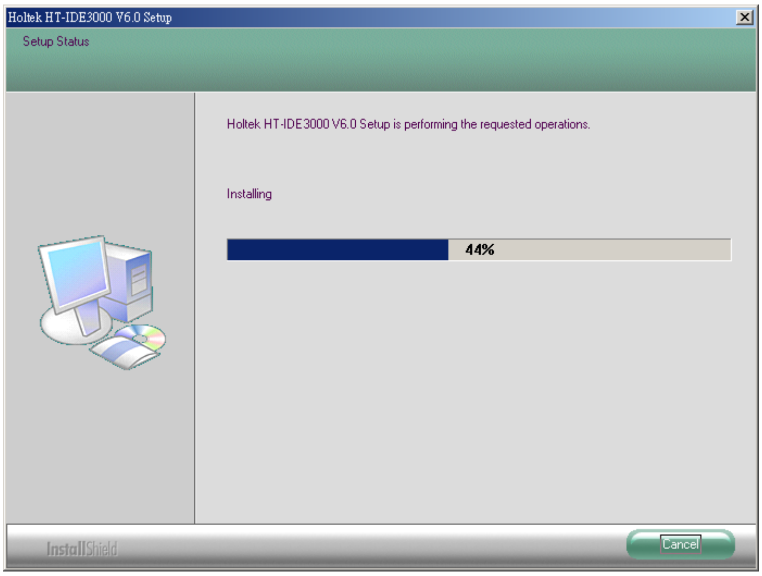


- 步骤 4

指定你希望安装 HT-IDE3000 的文件夹路径，然后按下<Next>按钮。

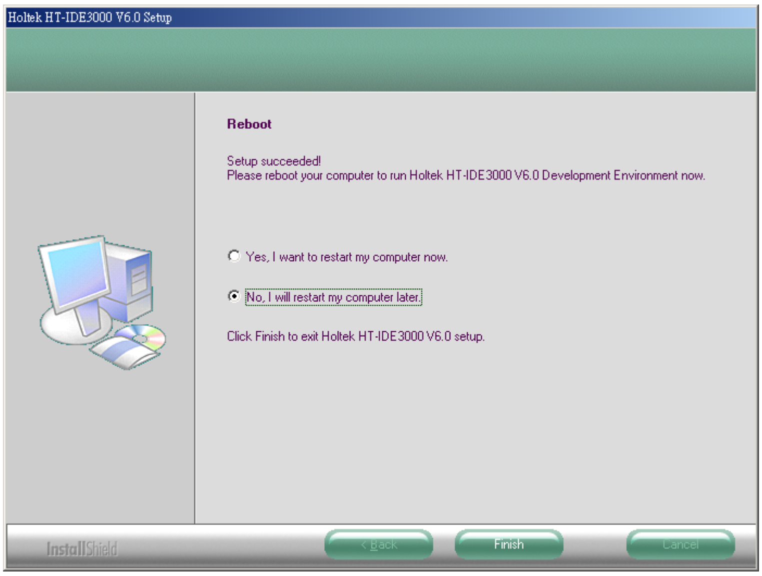
- 步骤 5

  SETUP 会将文件复制到你指定的文件夹。



- 步骤 6

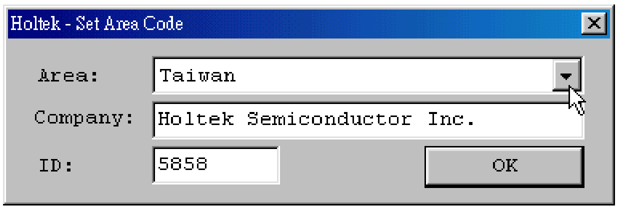
  安装成功的话，则会出现下面的对话框。



- 步骤 7

  请按下<Finish>按钮完成安装并重新启动计算机系统之后，你就可以执行 HT-IDE3000 程序了。SETUP 会在你指定的文件夹下，建立四个子文件夹(BIN、INCLUDE、LIB、SAMPLE)。BIN 子文件夹包含所有的系统可执行文件(EXE)、动态链接库(DLL)和配置文件(CFG、FMT)，INCLUDE 子文件夹包含所有由盛群所提供的包含文件(.H、.INC)，LIB 子文件夹包含由盛群所提供的库文件(.LIB)，SAMPLE 子文件夹包含范例程序。

  注意，在第一次执行 HT-IDE3000 之前，系统会要求输入如下图所示的公司资料，请选择适当的区域并填入公司名称及识别码，其中识别码可由 HT-IDE3000 的供应商提供。



第六章

快速开始

6

本章简述如何快速使用 HT-IDE3000 去开发一个应用程序项目。

步骤一:建立一个新项目

- 按 Project 菜单并选择 New 命令
- 输入项目名称并从组合框选择此项目使用的单片机型号
- 按下 OK 按键则系统将会要求设定单片机的掩膜选项
- 设定所有掩膜选项并按下 SAVE 键

步骤二:将源程序文件加到项目中

- 使用 File/New 命令建立源程序文件
- 撰写完程序后存盘，如 TEST.ASM 档名
- 按 Project 菜单并选择 Edit 命令
- 进入 Edit Project 对话框以便将源程序文件从项目中加入/删除
- 选择一个源程序文件，如 TEST.ASM，按下 Add 按钮
- 当所有源程序文件都被加入项目后，按下 OK 按钮

步骤三:建立项目

- 按 Project 菜单并选择 Build 命令
- 系统将会对项目中的所有源程序文件执行编译动作
  - 如果程序中有错误，只要在错误信息行连按两次，则系统将会提示错误发生的位置并且打开此错误所在的源程序文件，便可直接修改程序及存储文件
  - 如果所有程序文件都没有错误，则系统会产生一个执行文件(Task file)并且下载到 HT-ICE 中，准备仿真及除错
- 你可以重复此步骤直到没有错误

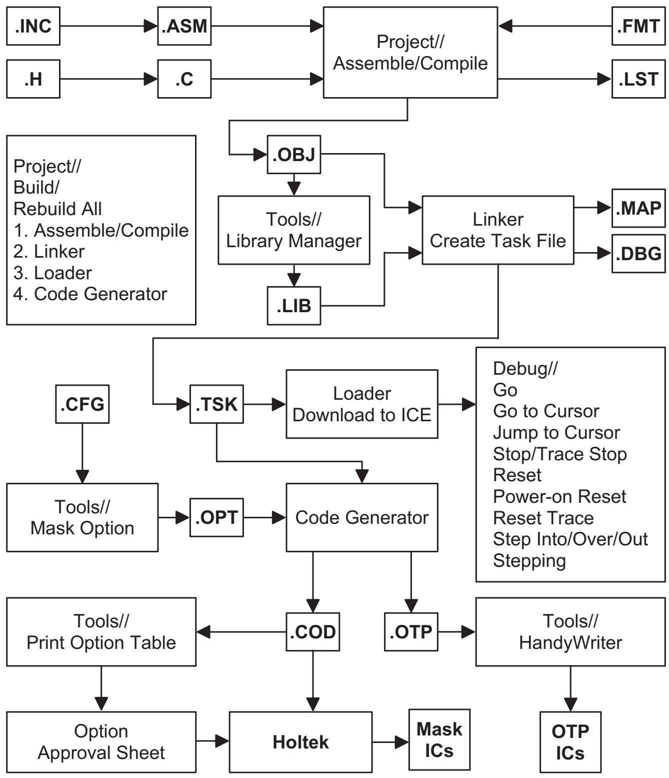
步骤四:烧录 OTP 单片机

- 建立项目，产生.OTP 文件
- 按下 Tools 菜单并选择 OTP Writer 命令去烧录 OTP 芯片

步骤五:传送程序与掩膜选项单至 Holtek

- 按 Project 菜单并选择 Print Option Table 命令去打印掩膜选项确认单
- 传送.COD 文件和掩膜选项确认单到盛群半导体公司，进行生产

程序及数据流程可由下图表示：



附录

附录 A

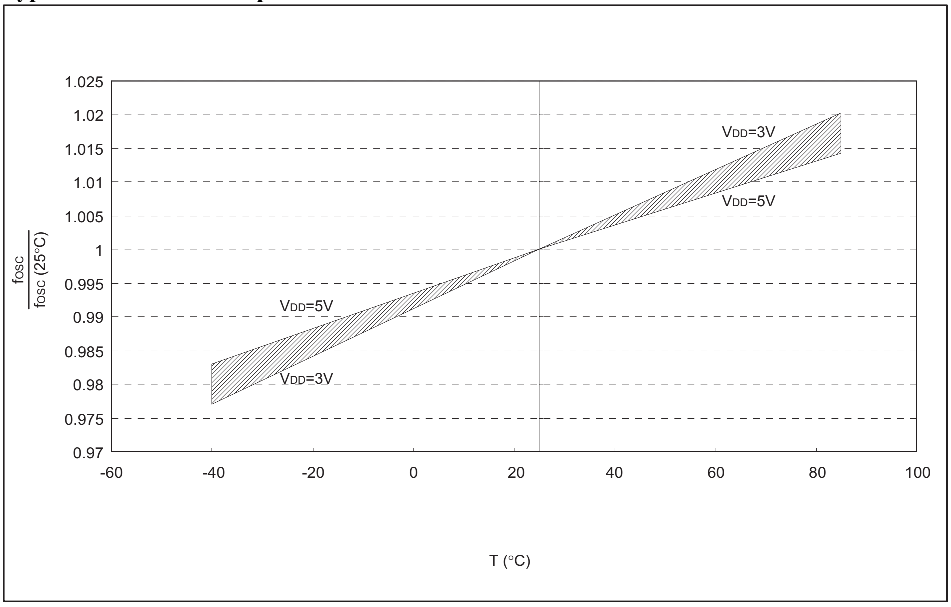
特性曲线图

A

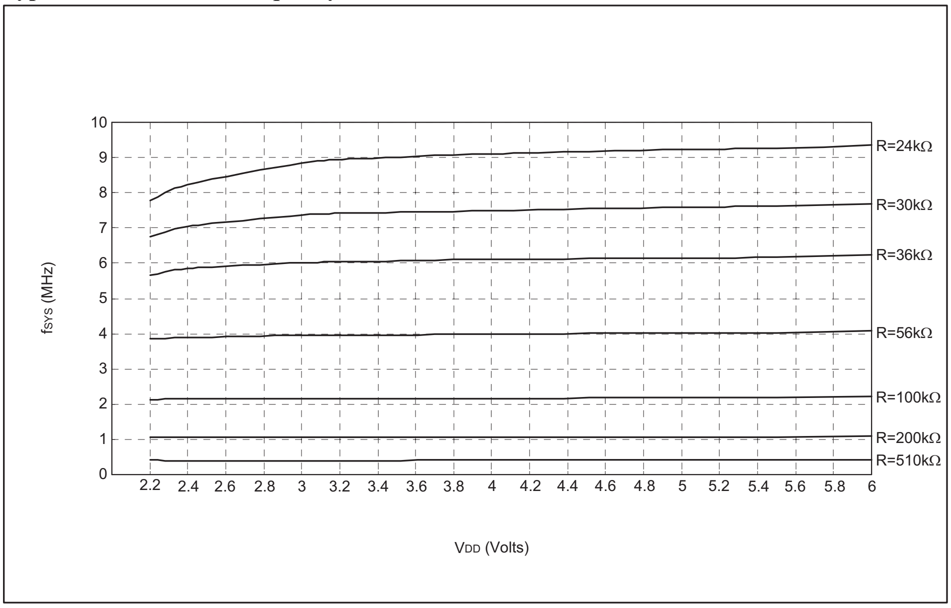
下面的特性曲线图描述典型的单片机行为特性，此处所显示的是在某一时期，测试不同批号的产品所收集到的统计数据，该信息只提供参考，且其特性图不是在生产过程中被测试。

在一些特性图中，超过操作范围的数据只是为显示信息之用，单片机只在规格范围内会正常操作。

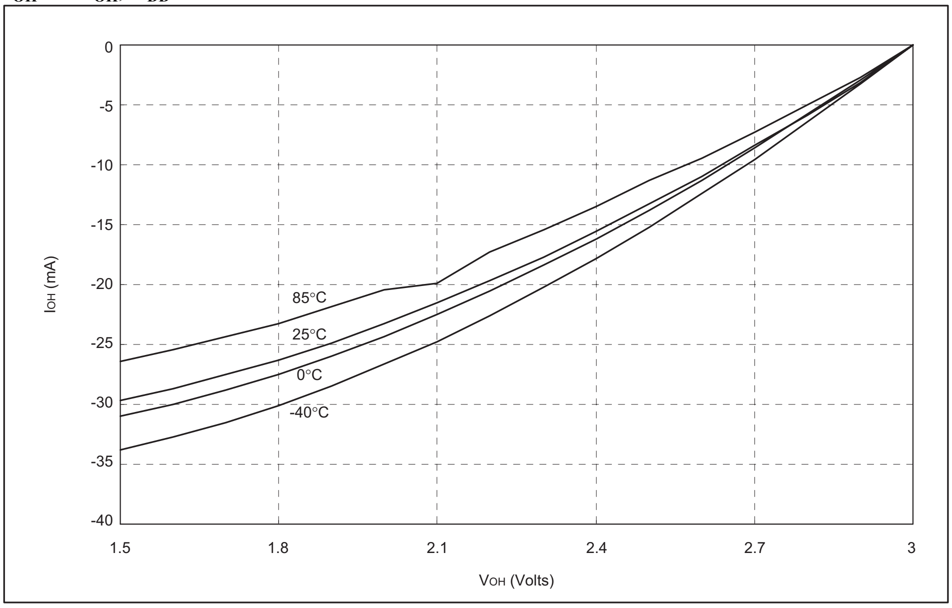
Typical RC OSC vs. Temperature



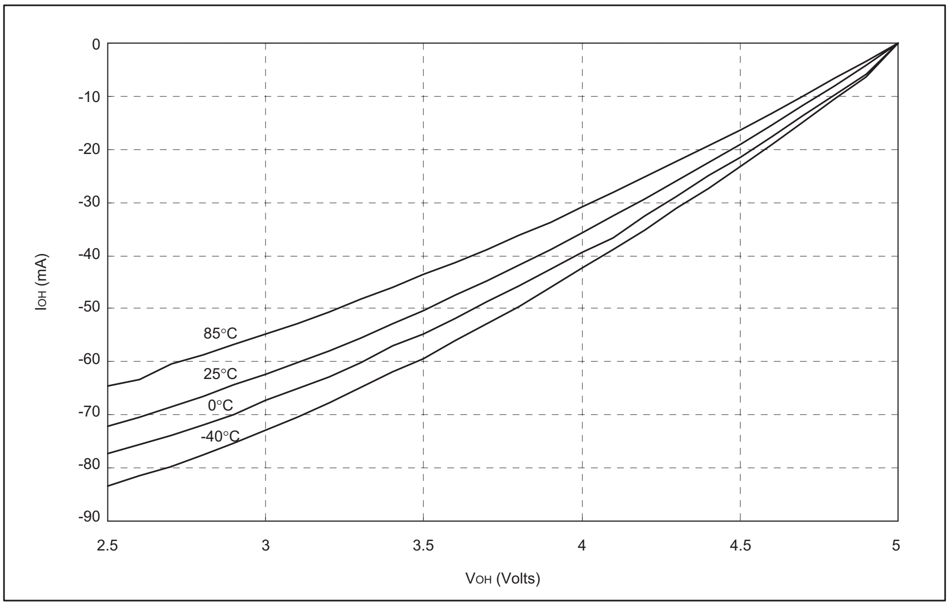
Typical RC Oscillator Frequency vs. V_{DD}



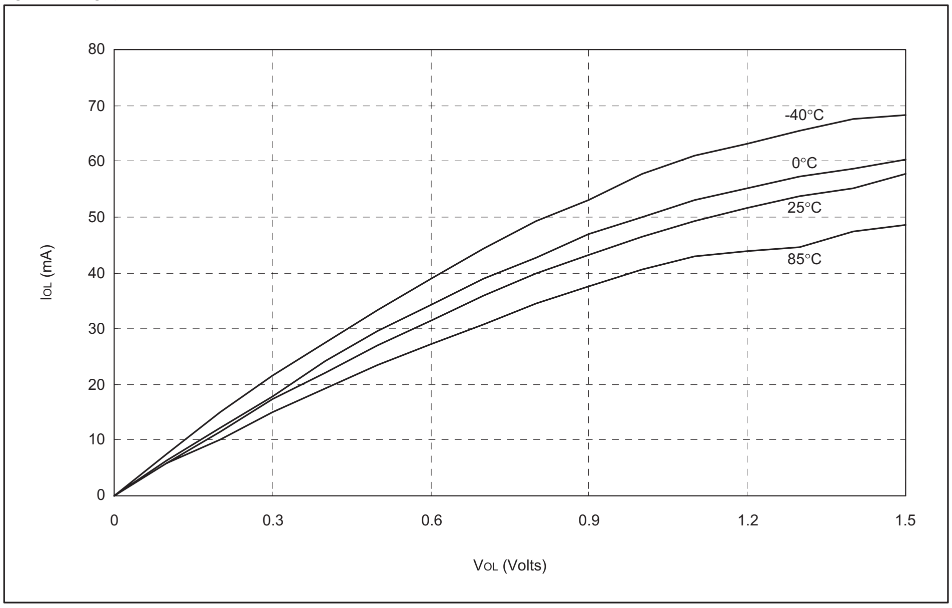
I_{OH} vs. V_{OH}, V_{DD}=3V



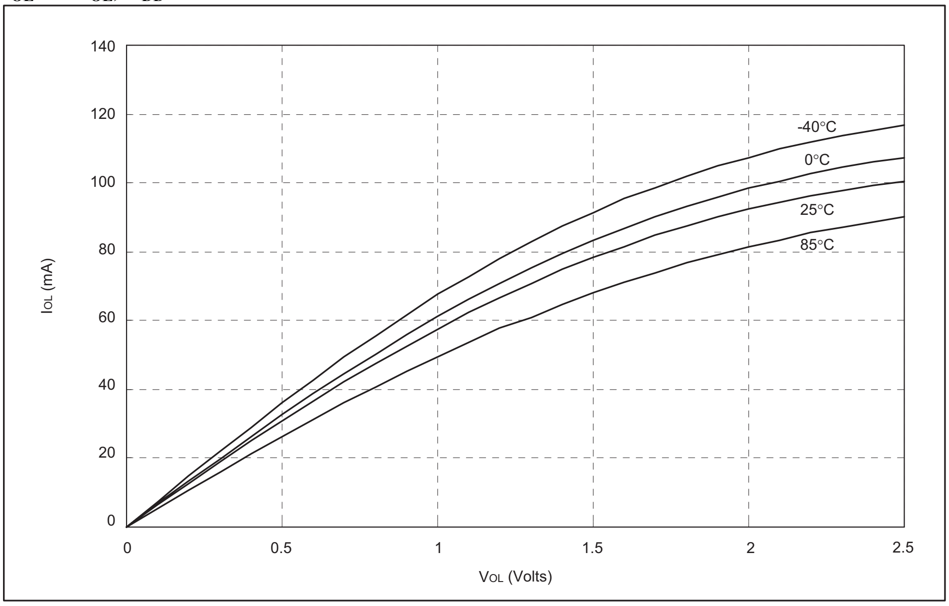
I_{OH} vs. V_{OH}, V_{DD}=5V



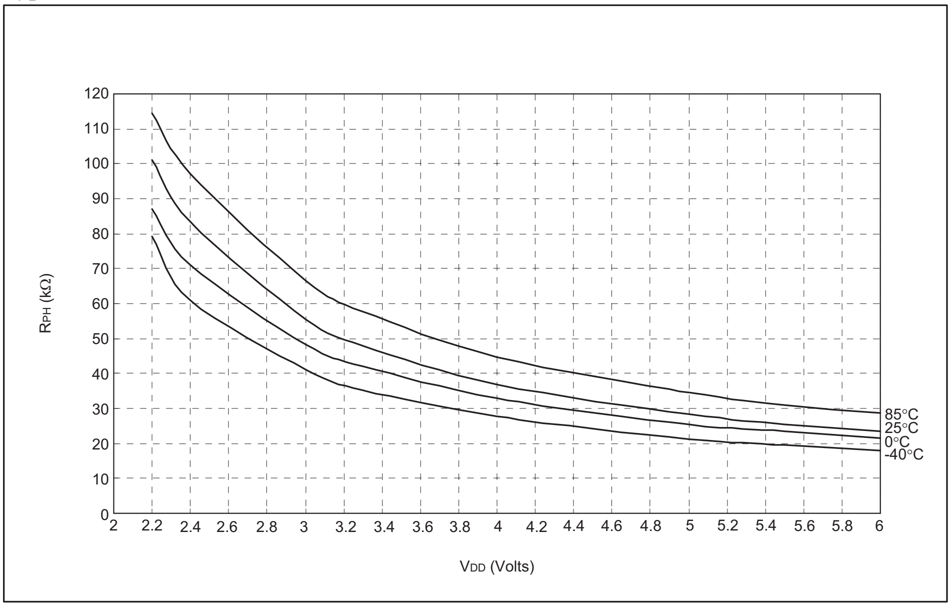
I_{OL} vs. V_{OL}, V_{DD}=3V



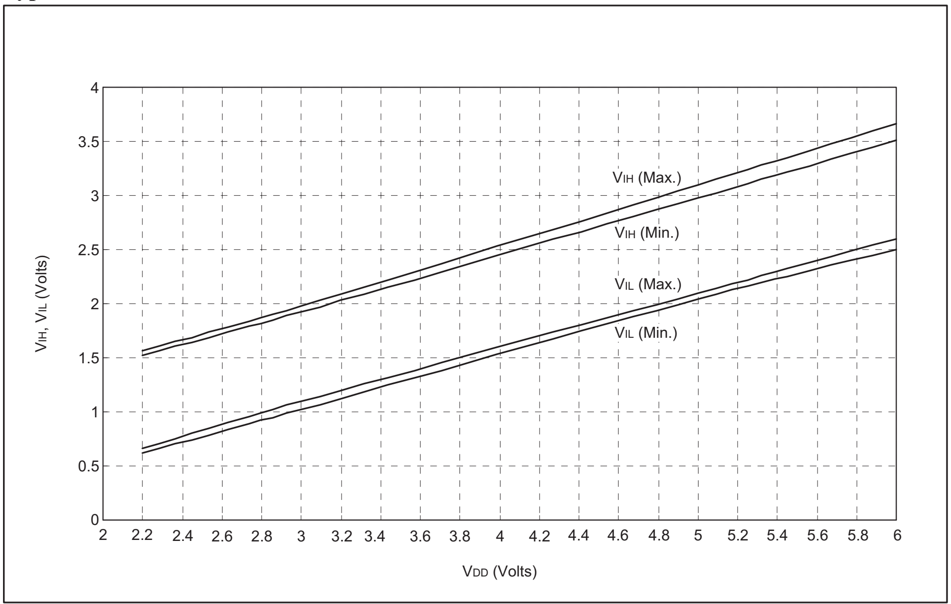
I_{OL} vs. V_{OL}, V_{DD}=5V



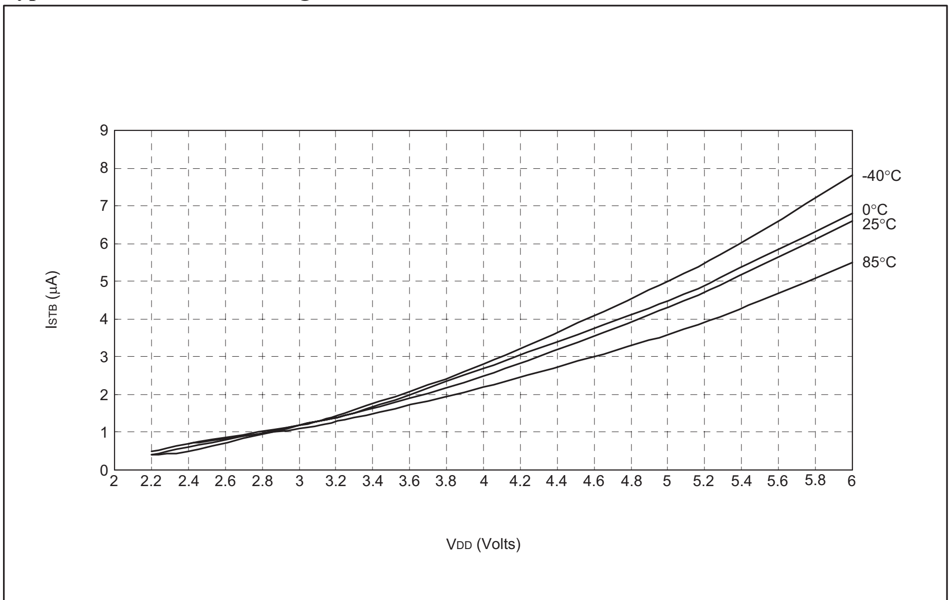
Typical R_{PH} vs. V_{DD}



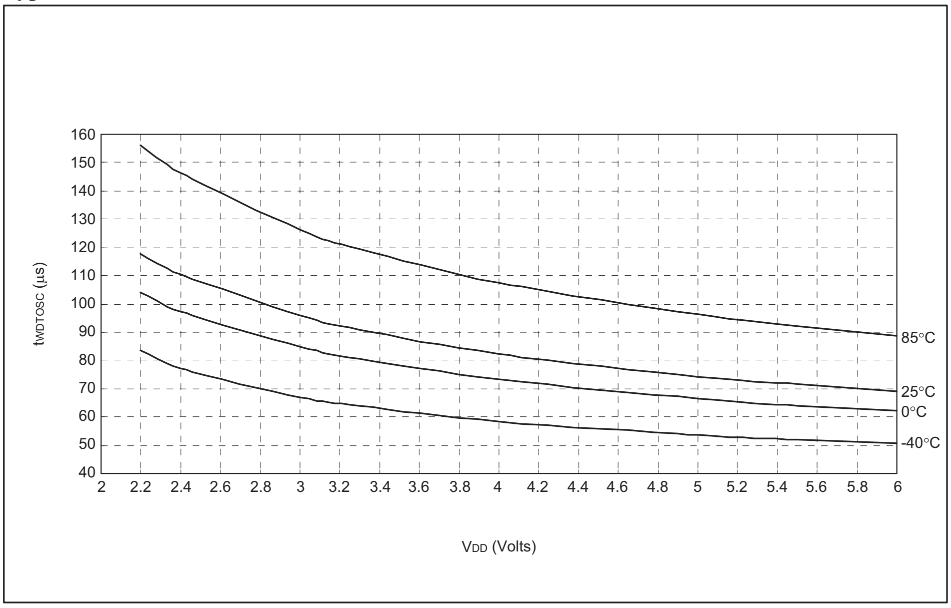
Typical V_{IH},V_{IL} vs. V_{DD} in -40°C to +85°C



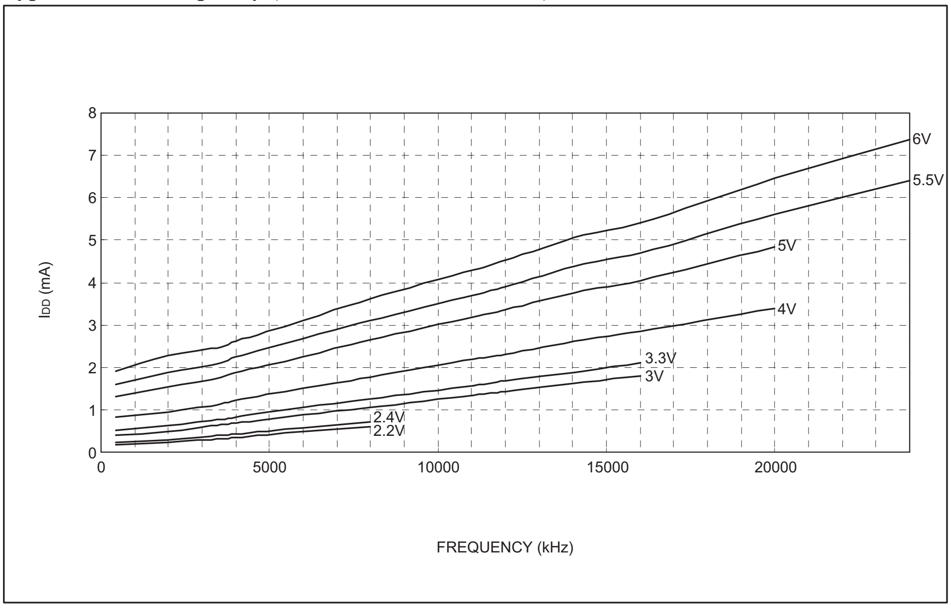
Typical I_{STB} vs. V_{DD} Watchdog Enable



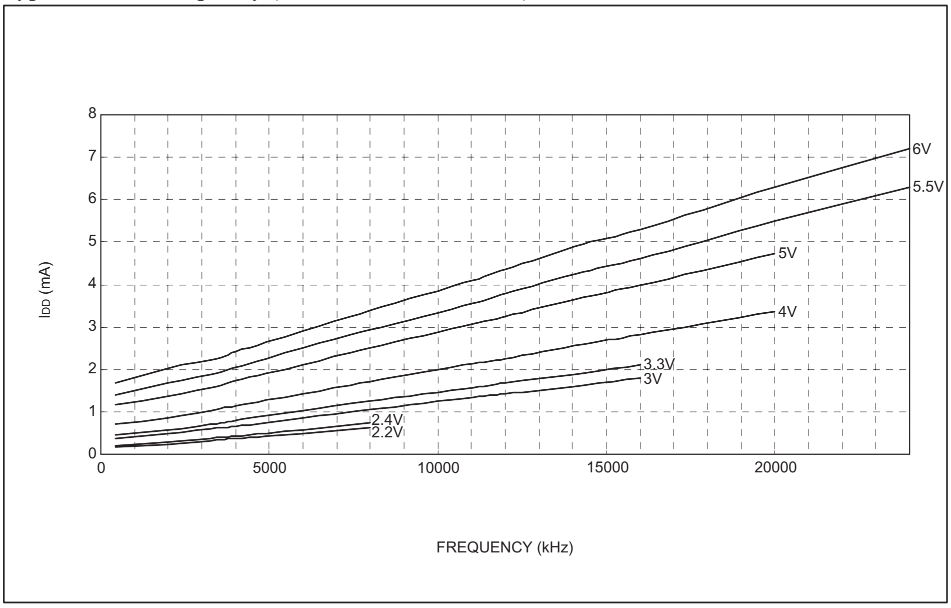
Typical t_{WDTOSC} vs. V_{DD}



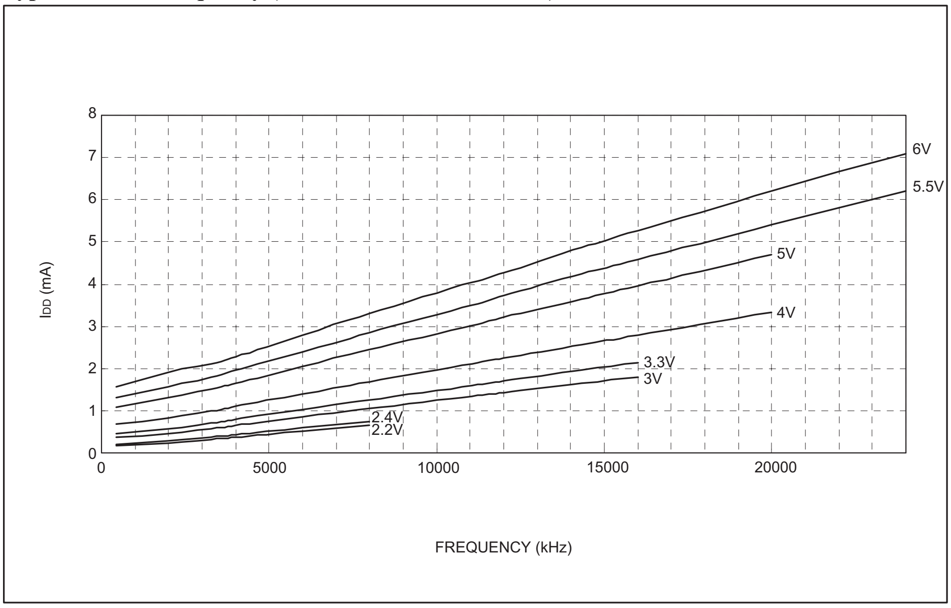
Typical I_{DD} vs. Frequency (External Clock, Ta=-40°C)



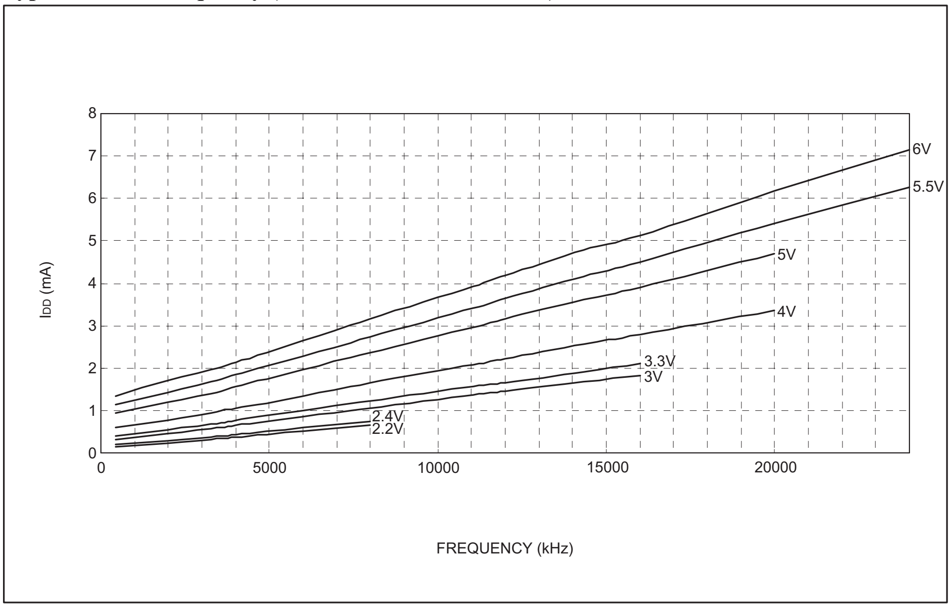
Typical I_{DD} vs. Frequency (External Clock, Ta=0°C)



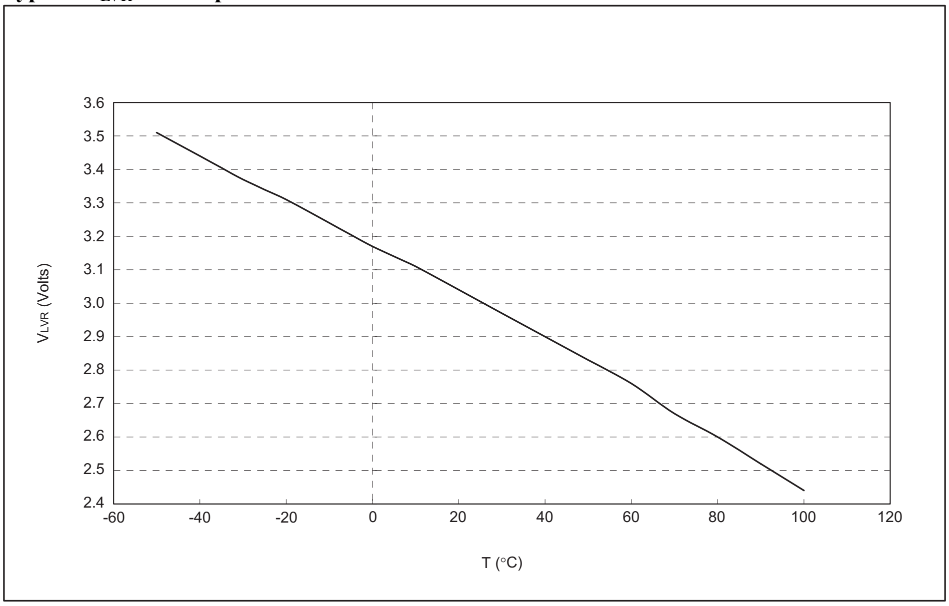
Typical I_{DD} vs. Frequency (External Clock, Ta=+25°C)



Typical I_{DD} vs. Frequency (External Clock, Ta=+85°C)



Typical V_{LVR} vs. Temperature

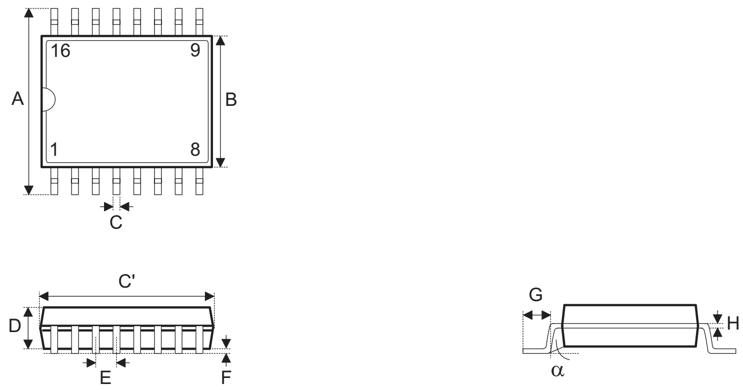


附录 **B**

封装信息

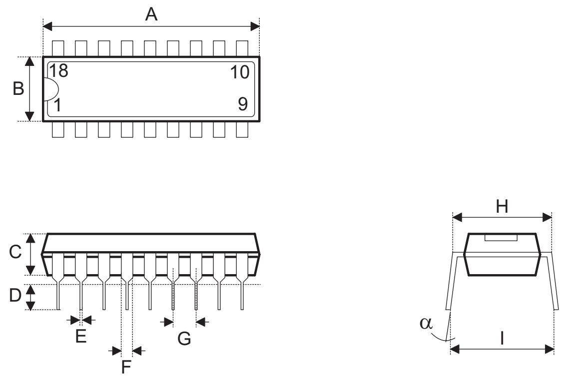
B

16-pin SSOP (150mil)外观尺寸



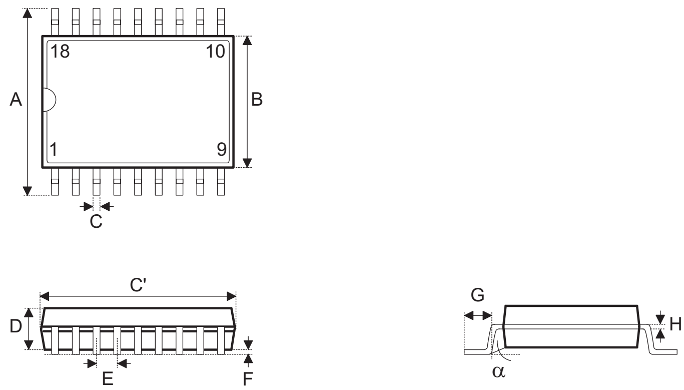
Symbol	Dimensions in mil		
	Min.	**Nom.**	**Max.**
A	228	—	244
B	150	—	157
C	8	—	12
C'	189	—	197
D	54	—	60
E	—	25	—
F	4	—	10
G	22	—	28
H	7	—	10
α	0°	—	8°

18-pin DIP (300mil)外观尺寸



Symbol	Dimensions in mil		
	Min.	**Nom.**	**Max.**
A	895	—	915
B	240	—	260
C	125	—	135
D	125	—	145
E	16	—	20
F	50	—	70
G	—	100	—
H	295	—	315
I	335	—	375
α	0°	—	15°

18-pin SOP (300mil)外观尺寸



Symbol	Dimensions in mil		
	Min.	**Nom.**	**Max.**
A	394	—	419
B	290	—	300
C	14	—	20
C'	447	—	460
D	92	—	104
E	—	50	—
F	4	—	—
G	32	—	38
H	4	—	12
α	0°	—	8°

盛群半导体股份有限公司（总公司）
台湾新竹市科学工业园区研新二路 3 号
电话:886-3-563-1999
传真:886-3-563-1189
网站:www.holtek.com.tw

盛群半导体股份有限公司（业务处）
台北市南港区园区街 3 之 2 号 4 楼之 2
电话: 886-2-2655-7070
传真: 886-2-2655-7373
传真: 886-2-2655-7383 (International sales hotline)

盛扬半导体（上海）有限公司
上海宜山路 889 号 2 号楼 7 楼200233
电话: 021-6485-5560
传真: 021-6485-0313
网站: www.holtek.com.cn

盛群半导体（香港）有限公司
香港九龙长沙湾道 777-779 号天安工业大厦 3 楼 A 座
电话:852-2-745-8288
传真:852-2-742-8657

Holmate Semiconductor, Inc.
46712 Fremont Blvd., Fremont, CA 94538
电话:510-252-9880
传真:510-252-9885
网站:www.holmate.com